INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT108-3-1-PYP

PYTHON PROGRAMMING

APU1F2303CS / APU1F2303EEE / APU1F2303IT / APU1F2303PE / APD1F2303CS / APD1F2303PE / APD1F2303CE / APD1F2303EEE / APD1F2303CGD / APU1F2303CS(DF) / APU1F2303MMT / APD1F2303CS(DF) / APU1F2303SE / APU1F2303CE / APU1F2303CGD / APD1F2303MMT

HAND OUT DATE: **10th MAY 2023**

HAND IN DATE: **7th JULY 2023**

WEIGHTAGE: **50%**

Table of Contents:

Grocery Store Inventory System

Introduction:

The inventory of a grocery shop is managed and tracked by a piece of software called the Grocery Inventory System. It provides a useful and efficient method for keeping track of various products, their quantities, prices, and other relevant information. Tools for managing inventory, taking stock, locating objects, and authenticating users are included in this system.

The objectives of the Grocery Inventory System are to automate processes, streamline the inventory management procedure, and provide accurate and up-to-date information on the product inventory of the grocery shop. Store owners and employees may easily add new products to the inventory, edit the details of already-existing items, delete items, take stock, look through replenishment lists, replenish stock, and conduct searches based on a number of criteria with the aid of this system.

Important traits:

1) User authentication: The system has user authentication to guarantee secure access. The user roles offered include administrators, inventory checkers, and purchasers, each with their own set of permissions and capabilities.

2) By inputting data such as the product code, description, category, unit, price, quantity, and minimum stock level, users are able to add new items to the inventory. It is also possible to update the specifics of already-existing objects to reflect changes.

3) If an item is no longer on hand or in stock at the grocery shop, it can be erased from the inventory.

4) Taking Stock: The system offers tools that allow users to change the quantity of items they have on hand. This feature makes it simpler to maintain accurate inventory records and identify items that need to be replaced.

5) Users can choose to create a replenishment list that features products whose stock levels are below the required minimum. This tool can be used by customers and business owners to identify products that require restocking.

6) Users can replace stock by changing the quantity of specific commodities thanks to technology. This function facilitates the process of maintaining appropriate inventory levels and restocking commodities.

7) A number of search parameters, such as description, code range, category, and price range, can be used to find items in the inventory. This feature enables easy retrieval of specific items from the inventory.

Administrative tasks: The further administrative functions of this component are only accessible to admin users. It carries out operations like adding new users, managing user accounts (such changing passwords), and maybe other administrative duties unique to the system.

management of files

The system uses file management to process and store data. Two text files are utilized:

1. inventory.txt: The code, description, category, unit, price, quantity, and minimum quantity of each item in the inventory are all listed in this text file.

2. userdata.txt: Keeps user information, such as username, password, and user type, in one place.

Categories and Permissions for Users:

User administration, inventory management, and the addition of new administrators are all completely in the authority of the administrator.

Inventory checker: Capable of taking an inventory and searching for items.

Purchaser: The one who is responsible of buying products for the inventory.

Logical Assumptions:

The code assumes the existence of three text files with the names "inventory_items.txt," "users.txt," and "replenish_list.txt".

The "inventory_items.txt" file contains the following details for each item in the inventory: code, description, category, unit, price, quantity, and minimum quantity.

The "users.txt" file lists each user account's username, password, and account type in that order, separated by commas.

The "replenish_list.txt" file contains the items that need to be replenished in the inventory.

The software assumes the user has authorized read-only and write access to the text files.

The "inventory_list" variable loads the inventory list as a list of lists from the "inventory_items.txt" file.

During the login process, the user must provide a username, password, and account type. The user's input is compared against the "users.txt" file to verify the credentials.

Following a successful login, the code presents the appropriate options and access privileges based on the user's account type.

The "admin" account type offers additional options when contrasted with the "inventory checker" and "purchaser" account types.

Creating new user accounts, adding items to the inventory, updating item information, deleting items from the inventory, taking stock of the inventory, identifying items that require restocking, restocking the inventory, searching for items in the inventory, and quitting the system are all abilities available to the "admin" user.

The "inventory checker" user has access to the inventory, may search for certain items, and can log out of the application.

The "purchaser" user has access to the system exit, inventory search, and products that require replenishment.

To determine whether the user wishes to add a new user account, use the "confirm()" method. It checks the user's selection and then moves forward appropriately.

The "add_user()" function is used to add a new user account to the "users.txt" file. When the account type is confirmed, the new account information is added to the file.

The "inventory_checker()" function presents alternatives to the "inventory checker" user and responds properly to their choice.

The "purchaser()" function gives the "purchaser" user options and carries out the selected course of action.

The "new_item()" function may be used by the "admin" user to add a new item to the inventory. The user's entry of the item data is then changed in the "inventory_items.txt" file and the "inventory_list" variable.

Pseudocode:

Pseudocode Functions:

```
PROGRAM Grocery Store Inventory System

FUNCTION insertItem():

    PRINT("Enter code of your item: ")
    APPEND codeAdd TO newItem
    PRINT("Enter description of your item: ")
    APPEND descriptionAdd TO newItem
    PRINT("Enter category of your item: ")
    APPEND categoryAdd TO newItem
    PRINT("Enter unit of your item: ")
    APPEND unitAdd TO newItem
    PRINT("Enter price of your item: ")
    APPEND priceAdd TO newItem
    PRINT("Enter quantity of your items: ")
    APPEND quantityAdd TO newItem
    PRINT("Enter minimum number of items allowed of this: ")
    APPEND minimumAdd TO newItem

END FUNCTION

FUNCTION updateItem():

    SET finalList TO lines[linesIndex]
    PRINT(finalList)
    SET question TO GET input("What would you like to change? ")
    SET ans TO GET input("What would you like to replace it with? ")
    SET changedList TO REPLACE(finalList, question, ans)
    PRINT(changedList)
    REMOVE lines[linesIndex] from lines
    INSERT changedList into lines at linesIndex
    PRINT(lines)

END FUNCTION
```

```
FUNCTION deleteItem():

    SET finalList TO lines[linesIndex]
    PRINT("This item has been deleted: " + finalList)
    REMOVE lines[linesIndex] from lines
    REMOVE lines[linesIndex] from lines
    PRINT(lines)

END FUNCTION

FUNCTION stockTaking():

    SET ask2 TO GET input("What would you like to change it to? ")
    REMOVE splitlist[5] from splitlist
    INSERT " '" + ask2 + "'" into splitlist[5]

END FUNCTION

FUNCTION viewReplenishList():

    PRINT("Code: {}".format(code))
    PRINT("Description: {}".format(description))
    PRINT("Category: {}".format(category))
    PRINT("Unit: {}".format(unit))
    PRINT("Price: {}".format(price))
    PRINT("Quantity: {}".format(quantity))
    PRINT("Minimum: {}".format(minimum))
    PRINT("~~~~~~~~~~~~~~~~~~~")

END FUNCTION

FUNCTION stockReplenish():

    ask3 = GET input("Enter new quantity: ")
    PRINT(GET item from splitList at index 5)
    replacedList = REPLACE item in finalList at index 5 with ask3
    PRINT(replacedList)
    REMOVE item at linesIndex from lines
    INSERT replacedList at linesIndex in lines

END FUNCTION
```

```
FUNCTION searchItems():

    IF ans is equal to "1" THEN
        ans2 = GET input("Enter description: ")
    ELSE IF ans is equal to "2" THEN
        ans2 = GET input("Enter a code range (ex. 3000, 5000): ")
        rangeCheck = True
        correctInput = False
    ELSE IF ans is equal to "3" THEN
        ans2 = GET input("Enter a category: ")
    ELSE IF ans is equal to "4" THEN
        ans2 = GET input("Enter a price range (ex. 30, 50): ")
        rangeCheck = True
        correctInput = False
    ELSE
        PRINT("Your entry was out of the range")
        correctInput = False
    END IF

END FUNCTION

FUNCTION adminFunction():

    IF choice is equal to "1" THEN
        insertItem()
    ELSE IF choice is equal to "2" THEN
        updateItem()
    ELSE IF choice is equal to "3" THEN
        deleteItem()
    ELSE IF choice is equal to "4" THEN
        stockTaking()
    ELSE IF choice is equal to "5" THEN
        viewReplenishList()
    ELSE IF choice is equal to "6" THEN
        stockReplenish()
    ELSE IF choice is equal to "7" THEN
        searchItems()
    ELSE IF choice is equal to "8" THEN

END FUNCTION
```

```
FUNCTION inventoryFunction():

    IF choice is equal to "1" THEN
        stockTaking()
    ELSE IF choice is equal to "2" THEN
        searchItems()
    ELSE IF choice is equal to "3" THEN
        PRINT("Exited system")
    END IF

END FUNCTION

FUNCTION purchaserFunction():

    IF choice is equal to "1" THEN
        viewReplenishList()
    ELSE IF choice is equal to "2" THEN
        stockReplenish()
    ELSE IF choice is equal to "3" THEN
        searchItems()
    ELSE IF choice is equal to "4" THEN
        PRINT("Exited system")
    END IF

END FUNCTION
```

```
FUNCTION loginauth():

    IF ask is equal to "Admin" THEN
        FOR x in adminList:
            IF correctPass is False THEN
                WHILE username is equal to x DO
                    index = FIND index of x in adminList
                    passIndex = index + 1
                    IF password is equal to adminList[passIndex] THEN
                        IF correctPass is False THEN
                            PRINT("Correct Username and Password")
                            correctPass = True
                            CALL adminFunction()
                    END IF
                END WHILE
            END IF

    ELSE IF ask is equal to "Inventory Checker" THEN
        FOR y in inventoryList:
            IF correctPass is False THEN
                IF username is equal to y THEN
                    index = FIND index of y in inventoryList
                    passIndex = index + 1
                    IF password is equal to inventoryList[passIndex] THEN
                        IF correctPass is False THEN
                            PRINT("Correct Username and Password")
                            correctPass = True
                            CALL inventoryFunction()
                    END IF
                END IF
            END IF
```

```
    ELSE IF ask is equal to "Purchaser" THEN
        FOR z in purchaserList:
            IF correctPass is False THEN
                IF username is equal to z THEN
                    index = FIND index of z in purchaserList
                    passIndex = index + 1
                    IF password is equal to purchaserList[passIndex] THEN
                        IF correctPass is False THEN
                            PRINT("Correct Username and Password")
                            correctPass = True
                            CALL purchaserFunction()
                    END IF
                END IF
            END IF

END FUNCTION
```

Pseudocode BEGIN:

```
BEGIN

    DECLARE newItem, finderIndex, lines, splitList AS LISTS
    DECLARE alreadyExists, used AS BOOLEANS
    DECLARE codeAdd, descriptionAdd, categoryAdd, unitAdd, priceAdd, quantityAdd, minimumAdd AS STRINGS
    DECLARE ask, ask2, question, ans, changedList, replacedList, finalList, Backtomenu AS STRINGS
    DECLARE finder, linesIndex, ask3 AS INTEGERS

    SET alreadyExists TO False

    WHILE alreadyExists EQUALS False:
        READ lines FROM "inventory.txt" into lines

        SET used TO False

        FOR x IN lines:
            SET finder TO FIND(x, codeAdd)
            APPEND finder TO finderIndex

            IF finder >= 1:
                SET alreadyExists TO False
                SET used TO True
                PRINT("This code is already being used, pick another one.")

        IF used EQUALS False:
            SET alreadyExists TO True

    CALL insertItem()-----------------------------------------------------

    SET finalList TO CONVERT newItem TO STRING

    OPEN "inventory.txt" in append mode as f
    WRITE newline character to f
    WRITE finalList to f

    SET Backtomenu TO GET input("Do you want to go back to the menu? (yes/no)")

    IF Backtomenu EQUALS "yes" THEN
        loginauth()
    ELSE
        PRINT("Ok")
```

```
"-------------------------------------------------------------------------------------"

    SET ask TO GET input("Enter product code of item you would like to update: ")
    READ lines FROM "inventory.txt" into lines

    FOR x IN lines:
        SET finder TO FIND(x, ask)
        APPEND finder TO finderIndex

        IF finder >= 1:
            SET linesIndex TO LENGTH(finderIndex) - 1

            CALL updateItem()-----------------------------------------------

            OPEN "inventory.txt" in write mode as file
            WRITE lines to file

    SET Backtomenu TO GET input("Do you want to go back to the menu? (yes/no)")

    IF Backtomenu EQUALS "yes" THEN
        CALL loginauth()
    ELSE
        PRINT("Ok")

"-------------------------------------------------------------------------------------"
```

```
    SET ask TO GET input("Enter Product Code of item you would like deleted: ")
    READ lines FROM "inventory.txt" into lines

    FOR x IN lines:
        SET finder TO FIND(x, ask)
        APPEND finder TO finderIndex

        IF finder >= 1:
            SET linesIndex TO LENGTH(finderIndex) - 1
            CALL deleteItem()-----------------------------------------------

            OPEN "inventory.txt" in write mode as file
            WRITE lines to file

    SET Backtomenu TO GET input("Do you want to go back to the menu? (yes/no)")

    IF Backtomenu EQUALS "yes" THEN
        loginauth()
    ELSE
        PRINT("Ok")

"-------------------------------------------------------------------------------------"
```

```
    SET ask TO GET input("Enter Product Code of item whose stock you would like updated: ")
    READ lines FROM "inventory.txt" into lines

    FOR x IN lines:
        SET finder TO FIND(x, ask)
        APPEND finder TO finderIndex

        IF finder >= 1:
            SET linesIndex TO LENGTH(finderIndex) - 1
            SET finalList TO lines[linesIndex]
            PRINT(finalList)

            SET splitlist TO SPLIT(finalList, ',')
            PRINT("Quantity of stock is " + splitlist[5])

            SET ans TO GET input("Would you like to change this quantity? (yes/no): ")

            IF ans EQUALS "yes" THEN
                CALL stockTaking()-----------------------------------------------------

                SET mySeparator TO ","
                SET y TO JOIN(splitlist, mySeparator)
                PRINT(y)

                REMOVE lines[linesIndex] from lines
                INSERT y into lines[linesIndex]

                OPEN "inventory.txt" in write mode as file
                WRITE lines to file

            ELSE IF ans EQUALS "no" THEN
                PRINT("Okay")

            ELSE:
                PRINT("That is not a valid input")

    SET Backtomenu TO GET input("Do you want to go back to the menu? (yes/no): ")

    IF Backtomenu EQUALS "yes" THEN
        loginauth()
    ELSE
        PRINT("Ok")
```

```
READ lines FROM "inventory.txt" into lines

PRINT("Replenish List:")
PRINT("~~~~~~~~~~~~~~~~~~~")

FOR x IN lines:
    SET item TO SPLIT(x.strip().strip("[]"), ",")
    SET item TO [value.strip().strip("'") for value IN item]

    IF LENGTH(item) >= 7:
        SET code TO item[0]
        SET description TO item[1]
        SET category TO item[2]
        SET unit TO item[3]
        SET price TO item[4]
        SET quantity TO CONVERT_TO_INTEGER(item[5])
        SET minimum TO CONVERT_TO_INTEGER(item[6])

        IF quantity < minimum THEN
            CALL viewReplenishList()-----------------------------------------------

SET Backtomenu TO GET input("Do you want to go back to the menu? (yes/no): ")

IF Backtomenu EQUALS "yes" THEN
    loginauth()
ELSE
    PRINT("Ok")
```

```
ask = GET input("Enter code of item you want replenished: ")
OPEN "inventory.txt" as f
READ lines from f
finderIndex = []
finder = -1

FOR x IN lines:
    finder = FIND x in ask
    APPEND finder to finderIndex
    IF finder >= 1 THEN
        linesIndex = GET length of finderIndex - 1
        finalList = GET item from lines at linesIndex
        splitList = STRIP finalList of "[],"
        splitList = STRIP each item in splitList of "'"
        PRINT(splitList)
        PRINT("Quantity is: " + GET item from splitList at index 5)
        ask2 = GET input("Would you like to update the quantity? (yes/no) ")

        IF ask2 is equal to "yes" THEN
            CALL stockReplenish()-----------------------------------------------
            OPEN "inventory.txt" as file
            WRITE lines to file
    END IF
END FOR

Backtomenu = GET input("Do you want to go back to menu? (yes/no)")
IF Backtomenu is equal to "yes" THEN
    loginauth()
ELSE
    PRINT("Ok")
END IF
```

```
PRINT("1 | Search item by description: ")
PRINT("2 | Search item by code range.")
PRINT("3 | Search items in a specific category")
PRINT("4 | Search items in a specific price range.")

ans = GET input("Enter a number from 1 to 4: ")
DECLARE correctInput, rangeCheck AS BOOLEAN
DECLARE ans2, firstInput, secondInput AS STRING

correctInput = True
rangeCheck = False

IF correctInput is equal to True THEN
    OPEN "inventory.txt" as f
    READ lines from f
    finderIndex = []
    finder = -1

    FOR x IN lines:
        finder = FIND x in ans2
        APPEND finder to finderIndex

        IF finder >= 1 THEN
            linesIndex = GET length of finderIndex - 1
            finalList = GET item from lines at linesIndex
            PRINT(finalList)
        END IF
    END FOR
END IF
```

```
IF rangeCheck is equal to True THEN
    OPEN "inventory.txt" as f
    READ lines from f
    splitAns = SPLIT ans2 by ", "
    firstInput = GET item from splitAns at index 0
    secondInput = GET item from splitAns at index 1
    finderIndex = []
    finder = -1

    FOR line IN lines:
        item = STRIP line of "[], "
        item = STRIP each item in item of "'"

        IF GET length of item >= 7 THEN
            code = GET item from item at index 0
            price = CONVERT GET item from item at index 4 to FLOAT

            IF ans is equal to "2" and firstInput <= code <= secondInput THEN
                PRINT(line)
            ELSE IF ans is equal to "4" and FLOAT(firstInput) <= price <= FLOAT(secondInput) THEN
                PRINT(line)
            END IF
        END IF
    END FOR
END IF

Backtomenu = GET input("Do you want to go back to menu? (yes/no)")

IF Backtomenu is equal to "yes" THEN
    loginauth()
ELSE
    PRINT("Ok")
END IF
```

```
PRINT("1 | Insert new Item")
PRINT("2 | Update Item")
PRINT("3 | Delete Item")
PRINT("4 | Stock Taking")
PRINT("5 | View Replenish List")
PRINT("6 | Stock Replenish")
PRINT("7 | Search Item")
PRINT("8 | Add new user")
PRINT("9 | Exit")

choice = GET input("Enter a number: ")

CALL adminFunction()------------------------------------------------------------
    PRINT("adding new user into userdata.txt")
    OPEN "userdata.txt" as h
    READ lines from h
    askUserType = GET input("Which user would you like to add? (admin, inventory checker, or purchaser)")
    askUsername = GET input("Enter username you would like to add: ")
    askPassword = GET input("Enter password you would like to add: ")
```

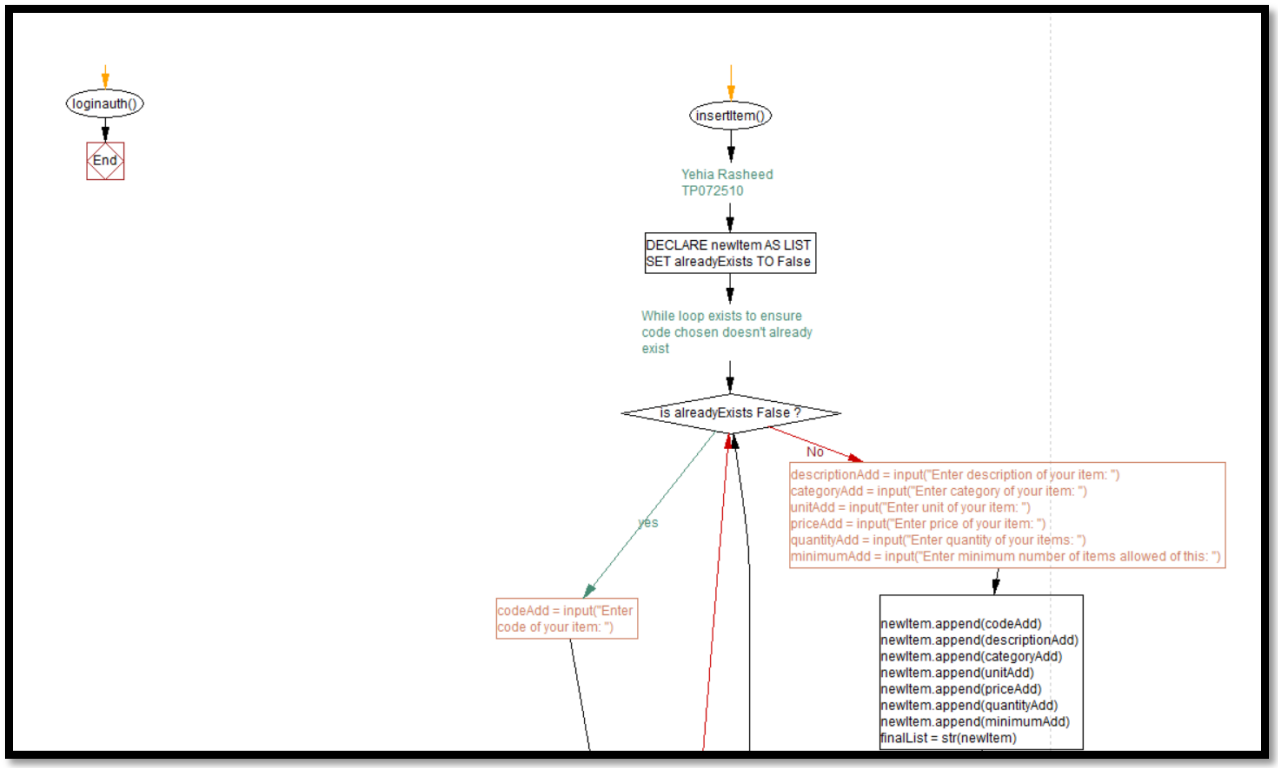
```
    IF askUserType is equal to "admin" THEN
        adminIndex = REPLACE lines[0] removing newline character
        splitAdmins = SPLIT adminIndex by ", "
        splitAdmins.append(askUsername)
        splitAdmins.append(askPassword + "\n")
        mySeparator = ", "
        x = JOIN splitAdmins with mySeparator
        PRINT(x)
        lines.pop(0)
        lines.insert(0, x)
        OPEN "userdata.txt" as k
        WRITE lines to k
    ELSE IF askUserType is equal to "inventory checker" THEN
        adminIndex = REPLACE lines[1] removing newline character
        splitAdmins = SPLIT adminIndex by ", "
        splitAdmins.append(askUsername)
        splitAdmins.append(askPassword + "\n")
        mySeparator = ", "
        x = JOIN splitAdmins with mySeparator
        PRINT(x)
        lines.pop(1)
        lines.insert(1, x)
        OPEN "userdata.txt" as k
        WRITE lines to k
    ELSE IF askUserType is equal to "purchaser" THEN
        adminIndex = REPLACE lines[2] removing newline character
        splitAdmins = SPLIT adminIndex by ", "
        splitAdmins.append(askUsername)
        splitAdmins.append(askPassword + "\n")
        mySeparator = ", "
        x = JOIN splitAdmins with mySeparator
        PRINT(x)
        lines.pop(2)
        lines.insert(2, x)
        OPEN "userdata.txt" as k
        WRITE lines to k
    ELSE
        PRINT("This user type does not exist")
    END IF
ELSE IF choice is equal to "9" THEN
    PRINT("Exited system")
END IF
```

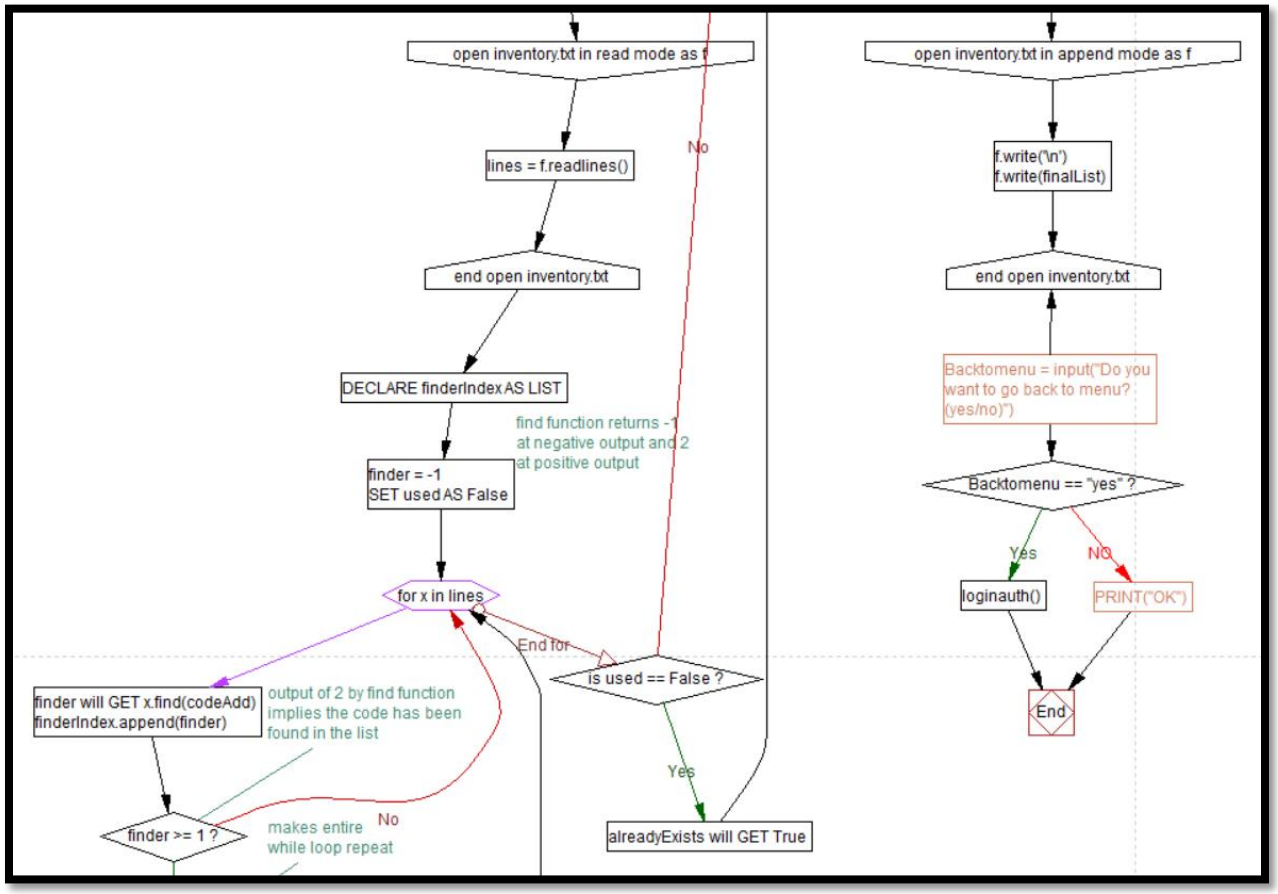
```
PRINT("You can do stock taking and searching items")
PRINT("1 | Stock taking")
PRINT("2 | Searching items")
PRINT("3 | Exit")

choice = GET input("Enter a number: ")

CALL inventoryFunction()--------------------------------------------------
```

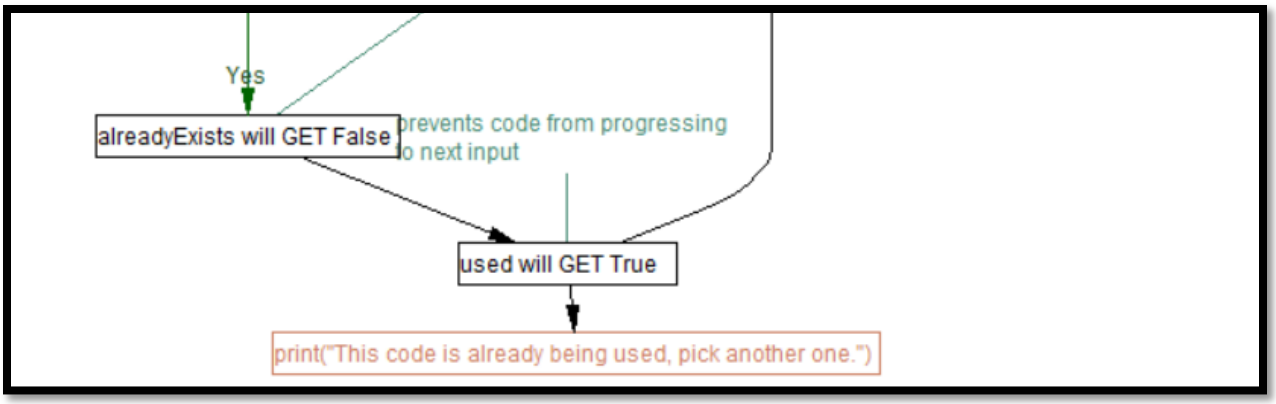
```
PRINT("View replenish list, stock replacement, and search items")
PRINT("1 | Replenish List")
PRINT("2 | Stock Replacement")
PRINT("3 | Search Items")
PRINT("4 | Exit")

choice = GET input("Enter a number: ")

CALL purchaserFunction()--------------------------------------------------
```

FlowChart Design:

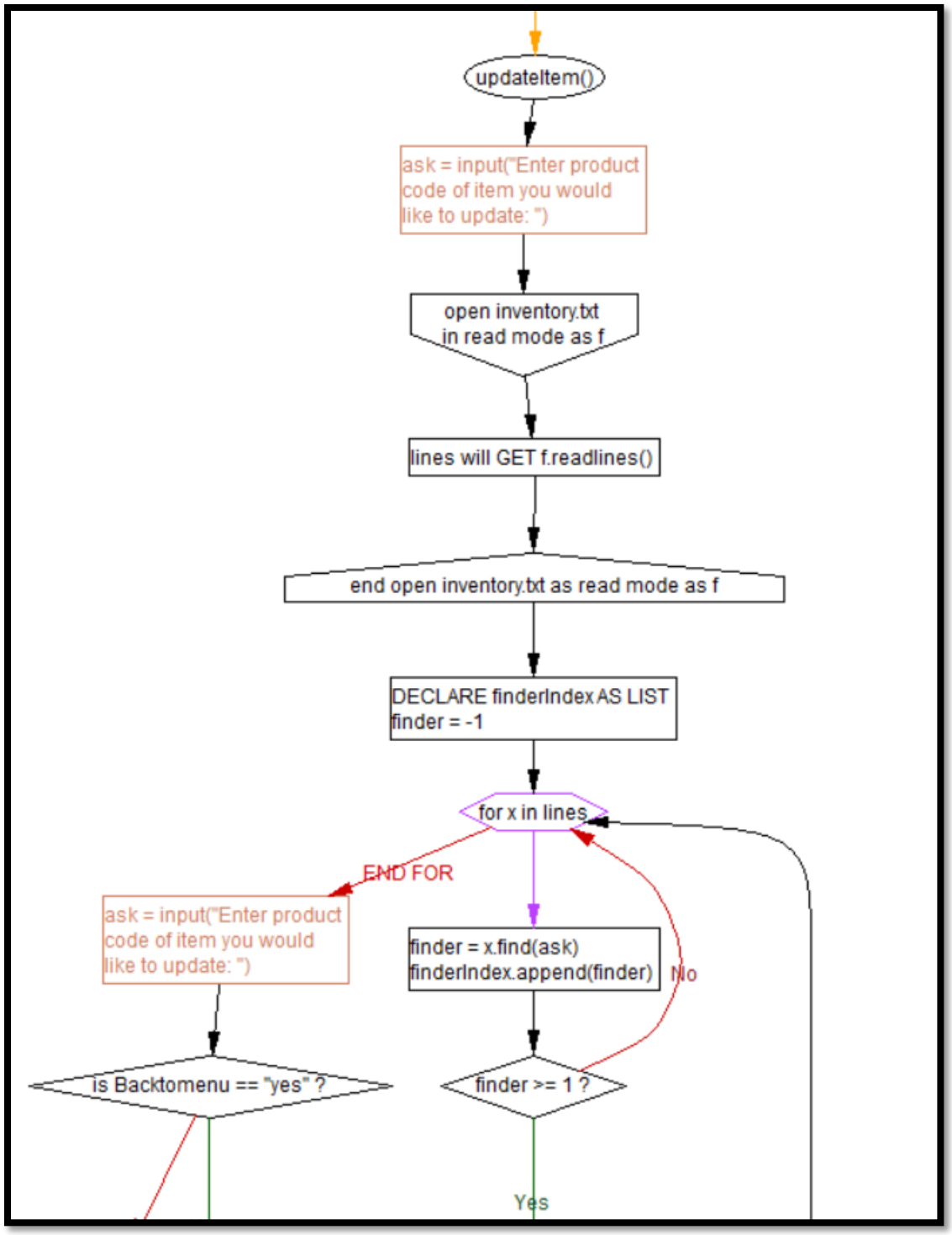
Flowchart Insert Item Function:

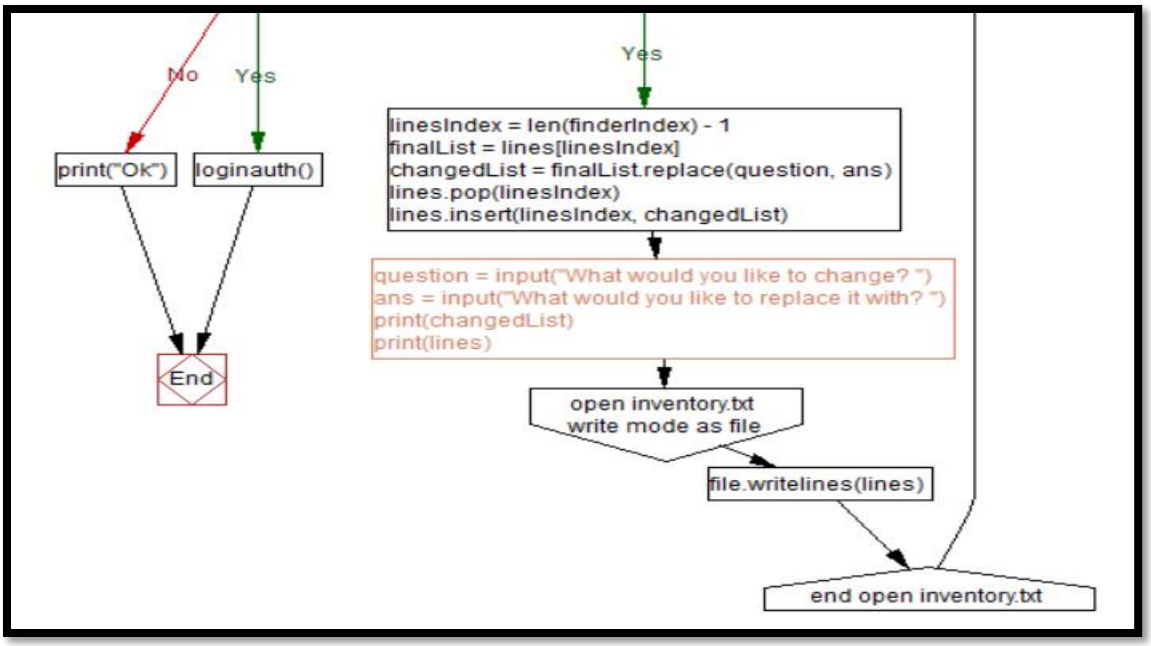




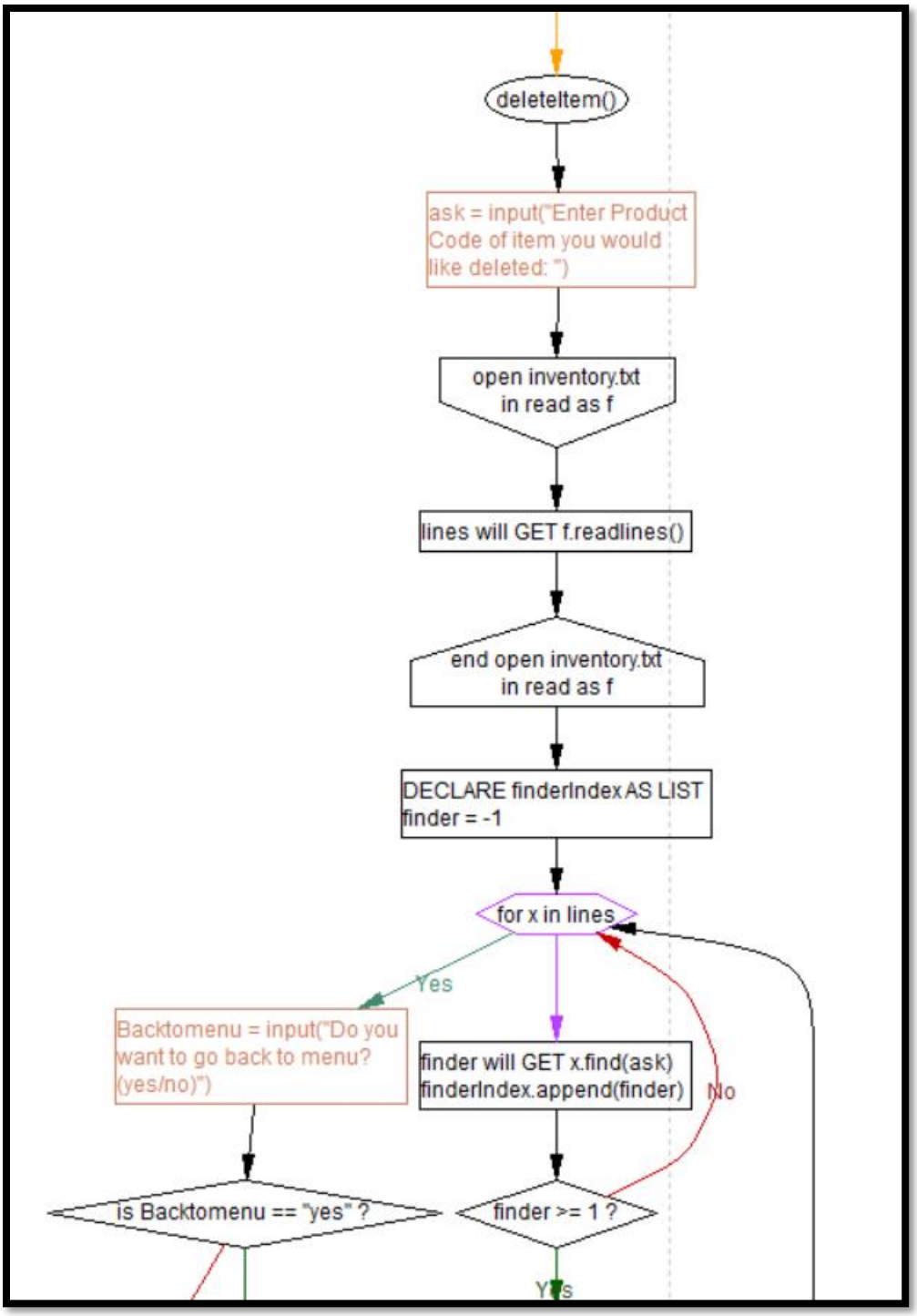

Yes
alreadyExists will GET False
prevents code from progressing
o next input
used will GET True
print("This code is already being used, pick another one.")

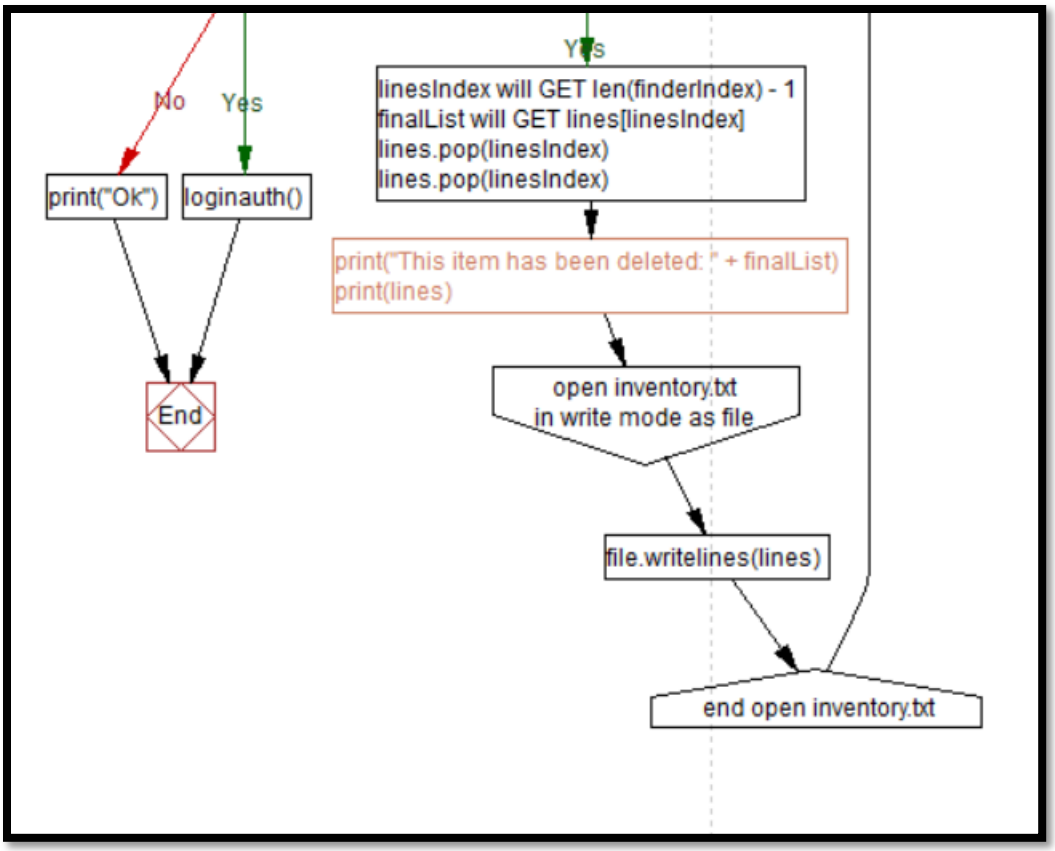
Flowchart Update Item Function:



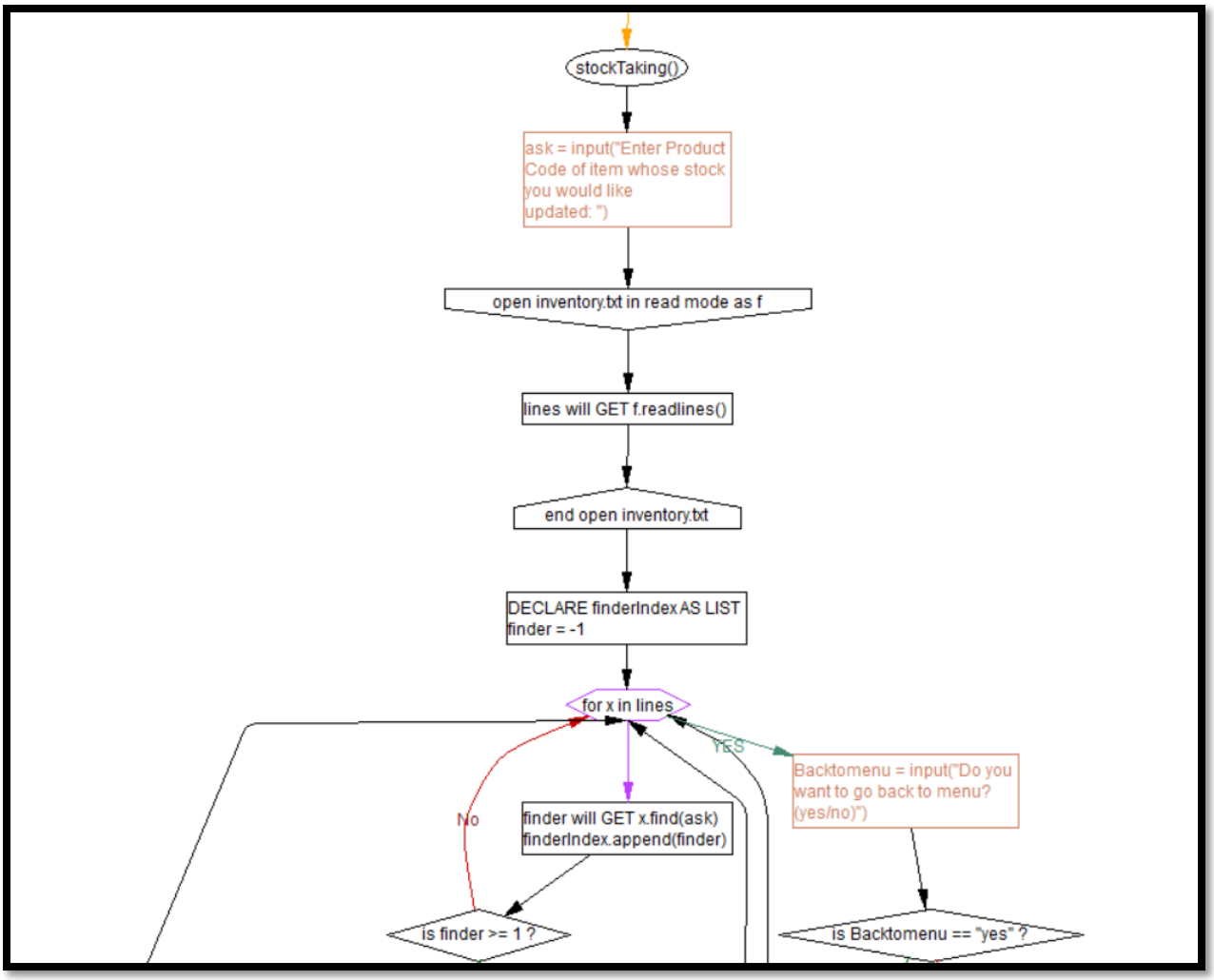

No
Yes
print("Ok")
loginauth()
End
Yes
linesIndex = len(finderIndex) - 1
finalList = lines[linesIndex]
changedList = finalList.replace(question, ans)
lines.pop(linesIndex)
lines.insert(linesIndex, changedList)
question = input("What would you like to change? ")
ans = input("What would you like to replace it with? ")
print(changedList)
print(lines)
open inventory.txt
write mode as file
file.writelines(lines)
end open inventory.txt

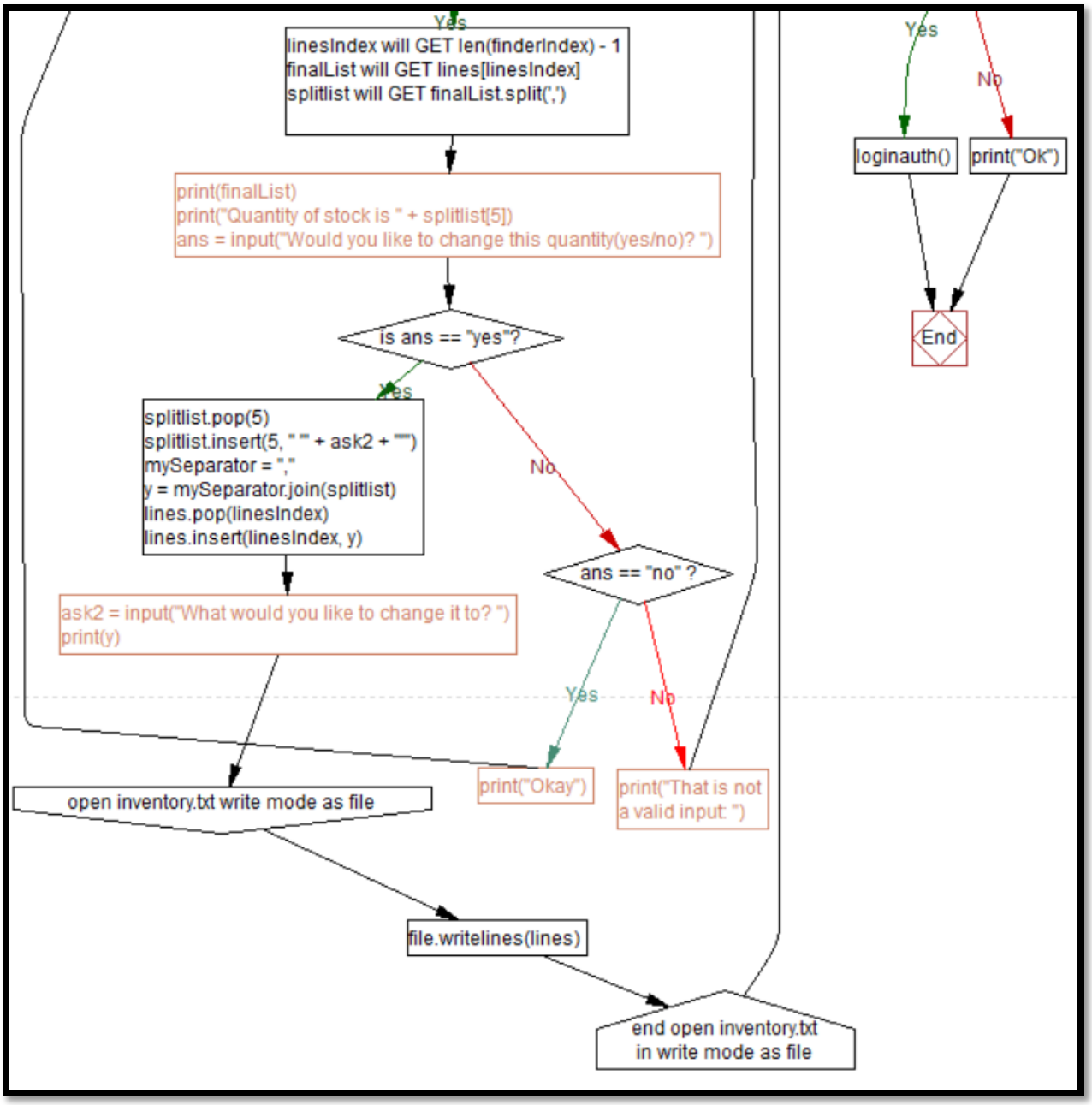
Flowchart Delete Item Function:



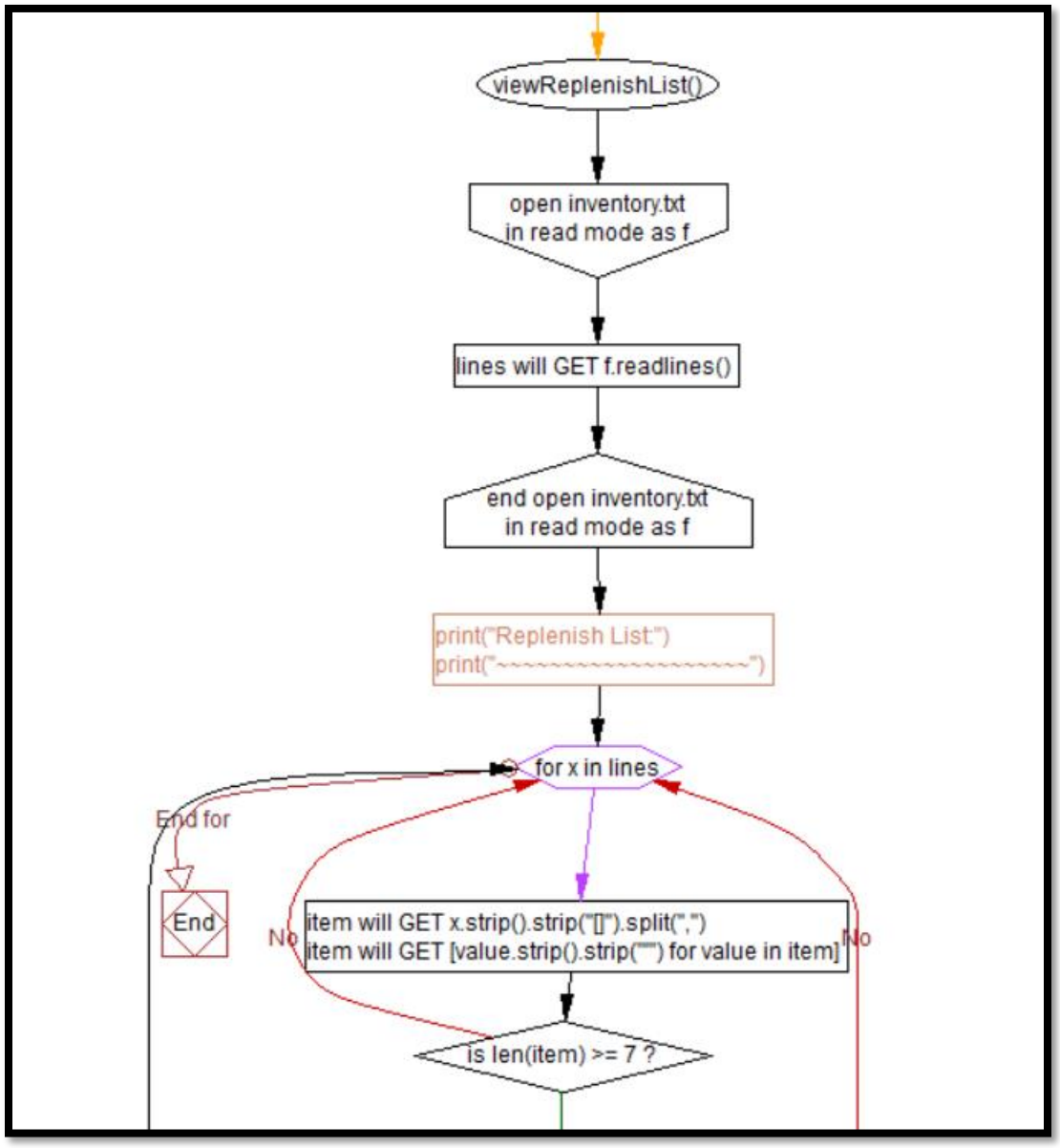

No
Yes
print("Ok")
loginauth()
End
Yes
linesIndex will GET len(finderIndex) - 1
finalList will GET lines[linesIndex]
lines.pop(linesIndex)
lines.pop(linesIndex)
print("This item has been deleted: " + finalList)
print(lines)
open inventory.txt
in write mode as file
file.writelines(lines)
end open inventory.txt

Flowchart Stock Taking Function:

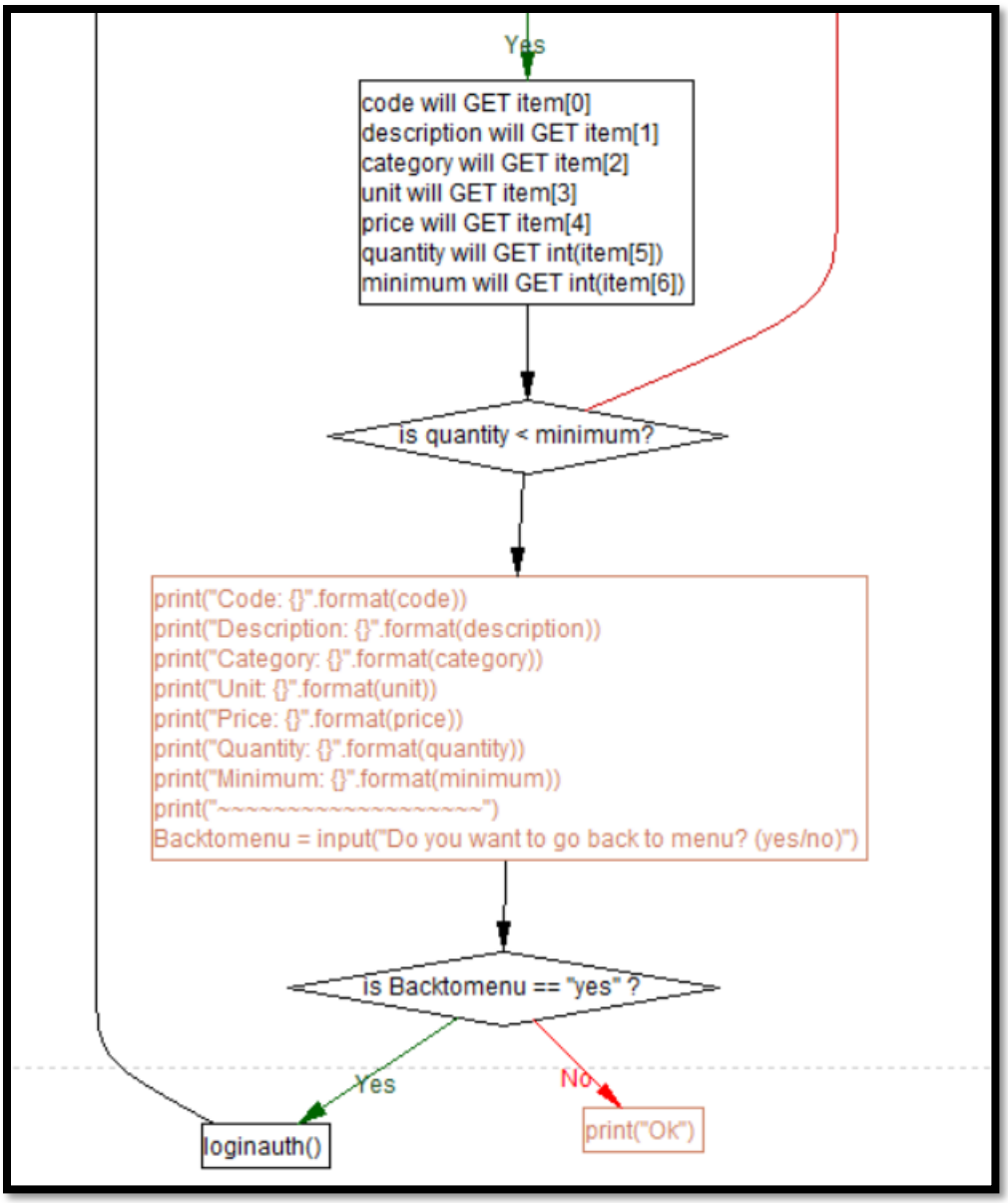

stockTaking()
ask = input("Enter Product Code of item whose stock you would like updated: ")
open inventory.txt in read mode as f
lines will GET f.readlines()
end open inventory.txt
DECLARE finderIndex AS LIST
finder = -1
for x in lines
YES
Backtomenu = input("Do you want to go back to menu? (yes/no)")
No
finder will GET x.find(ask)
finderIndex.append(finder)
is finder >= 1 ?
is Backtomenu == "yes" ?


Yes
linesIndex will GET len(finderIndex) - 1
finalList will GET lines[linesIndex]
splitlist will GET finalList.split(',')
print(finalList)
print("Quantity of stock is " + splitlist[5])
ans = input("Would you like to change this quantity(yes/no)? ")
is ans == "yes"?
Yes
splitlist.pop(5)
splitlist.insert(5, " "" + ask2 + """)
mySeparator = ","
y = mySeparator.join(splitlist)
lines.pop(linesIndex)
lines.insert(linesIndex, y)
No
ans == "no" ?
ask2 = input("What would you like to change it to? ")
print(y)
Yes
No
print("Okay")
print("That is not a valid input: ")
open inventory.txt write mode as file
file.writelines(lines)
end open inventory.txt in write mode as file
Yes
No
loginauth()
print("Ok")
End

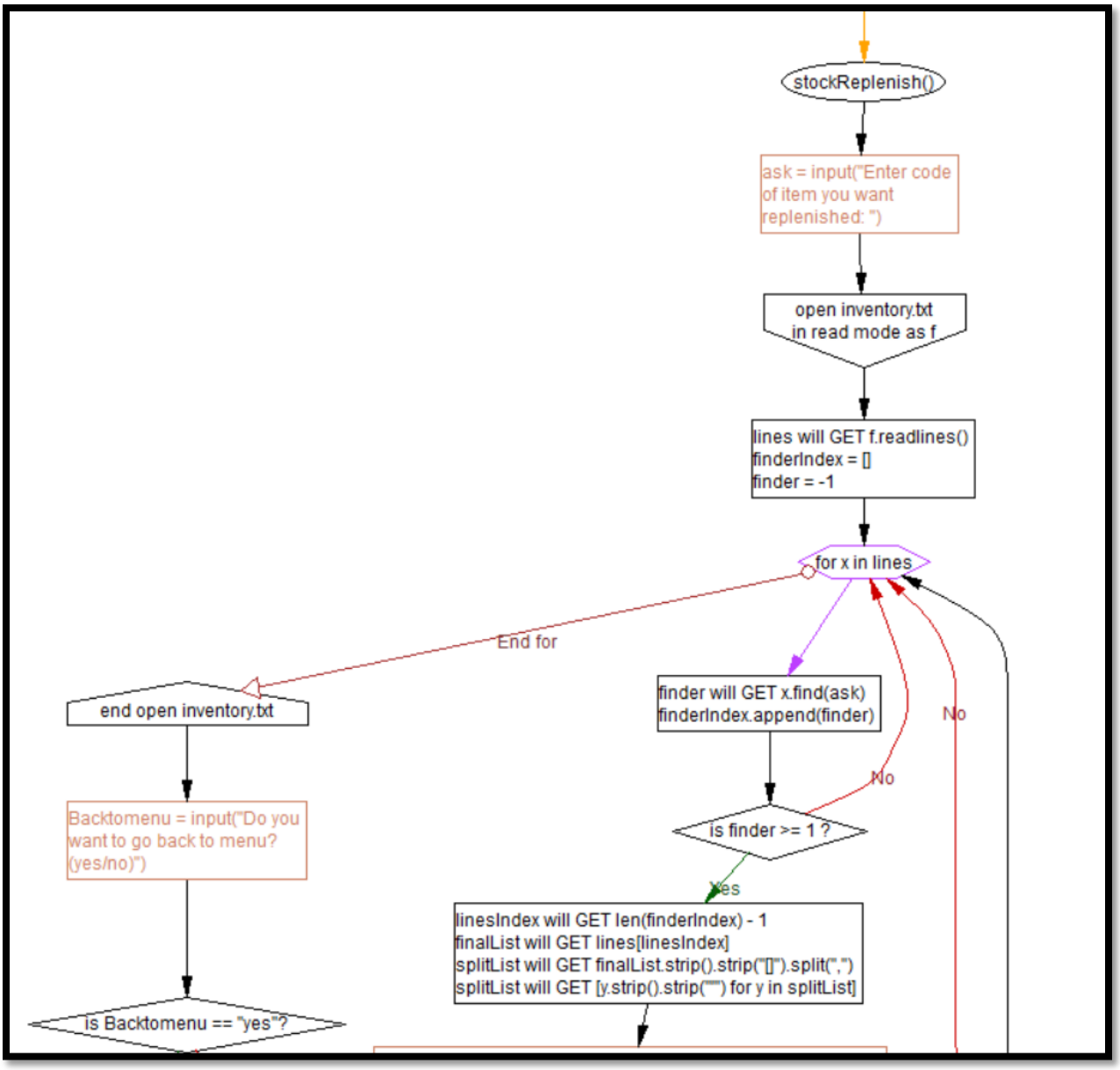
Flowchart View Replenish List:

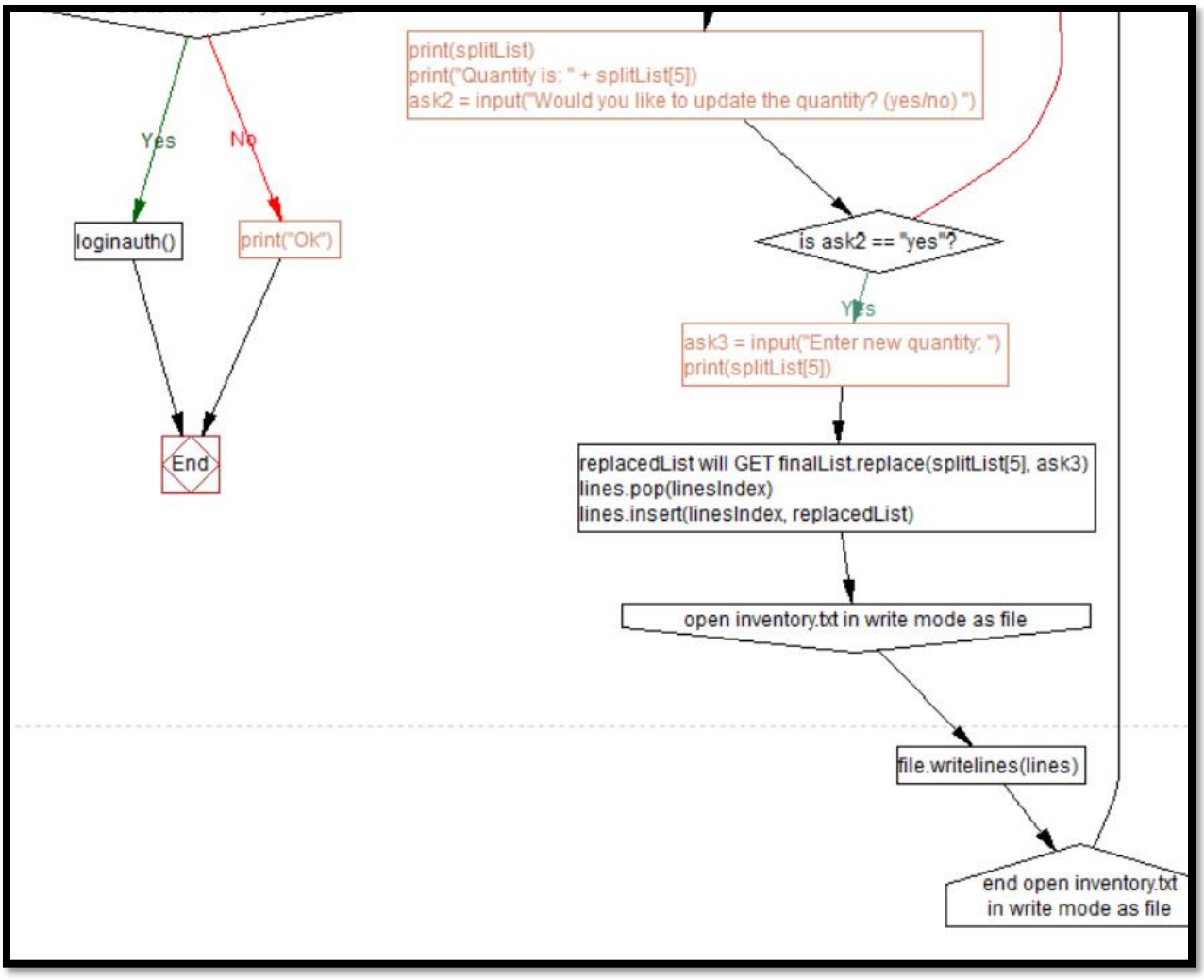


:

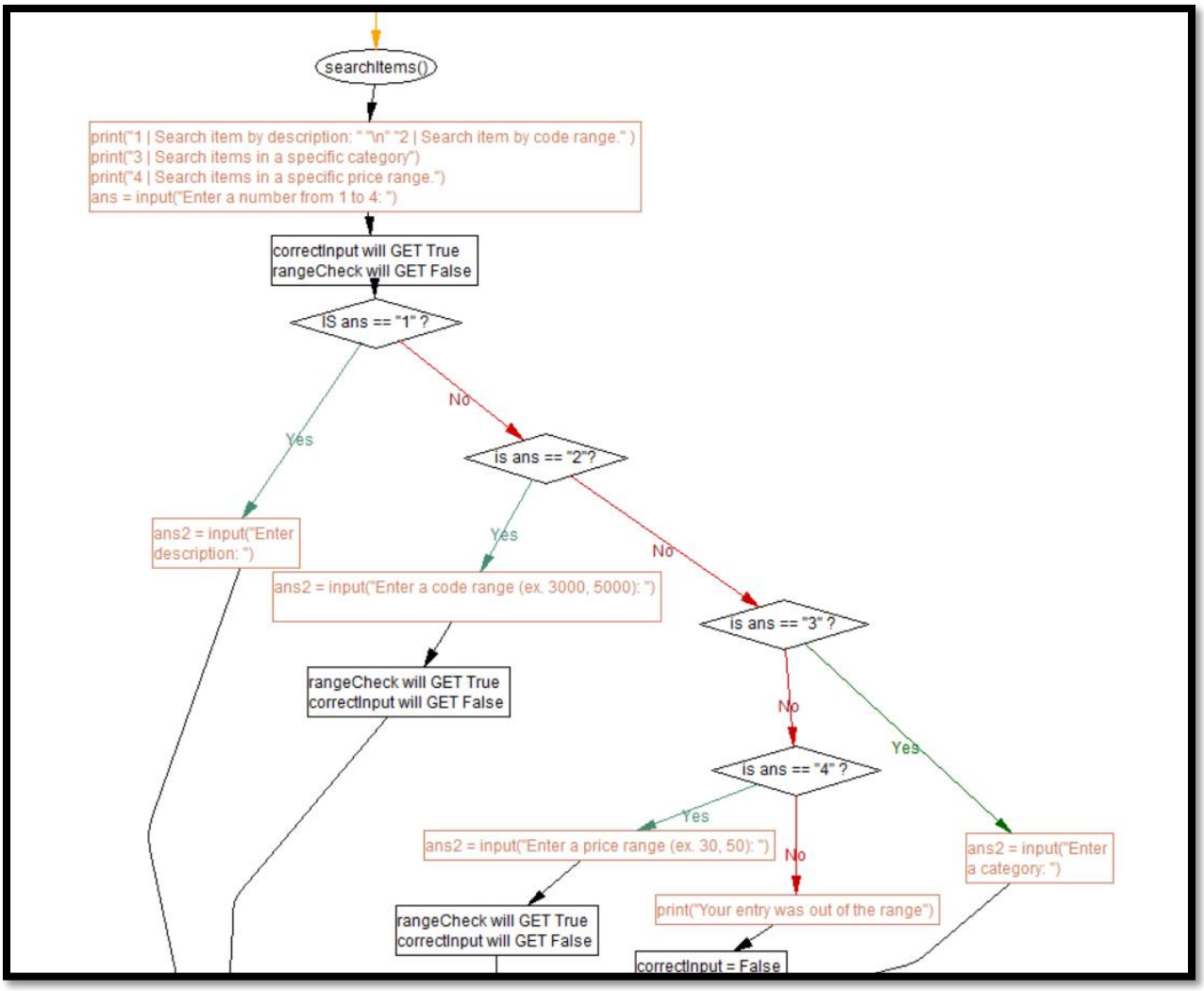

Yes
code will GET item[0]
description will GET item[1]
category will GET item[2]
unit will GET item[3]
price will GET item[4]
quantity will GET int(item[5])
minimum will GET int(item[6])
is quantity < minimum?
print("Code: {}".format(code))
print("Description: {}".format(description))
print("Category: {}".format(category))
print("Unit: {}".format(unit))
print("Price: {}".format(price))
print("Quantity: {}".format(quantity))
print("Minimum: {}".format(minimum))
print("~~~~~~~~~~~~~~~~~~~~")
Backtomenu = input("Do you want to go back to menu? (yes/no)")
is Backtomenu == "yes" ?
Yes
No
loginauth()
print("Ok")

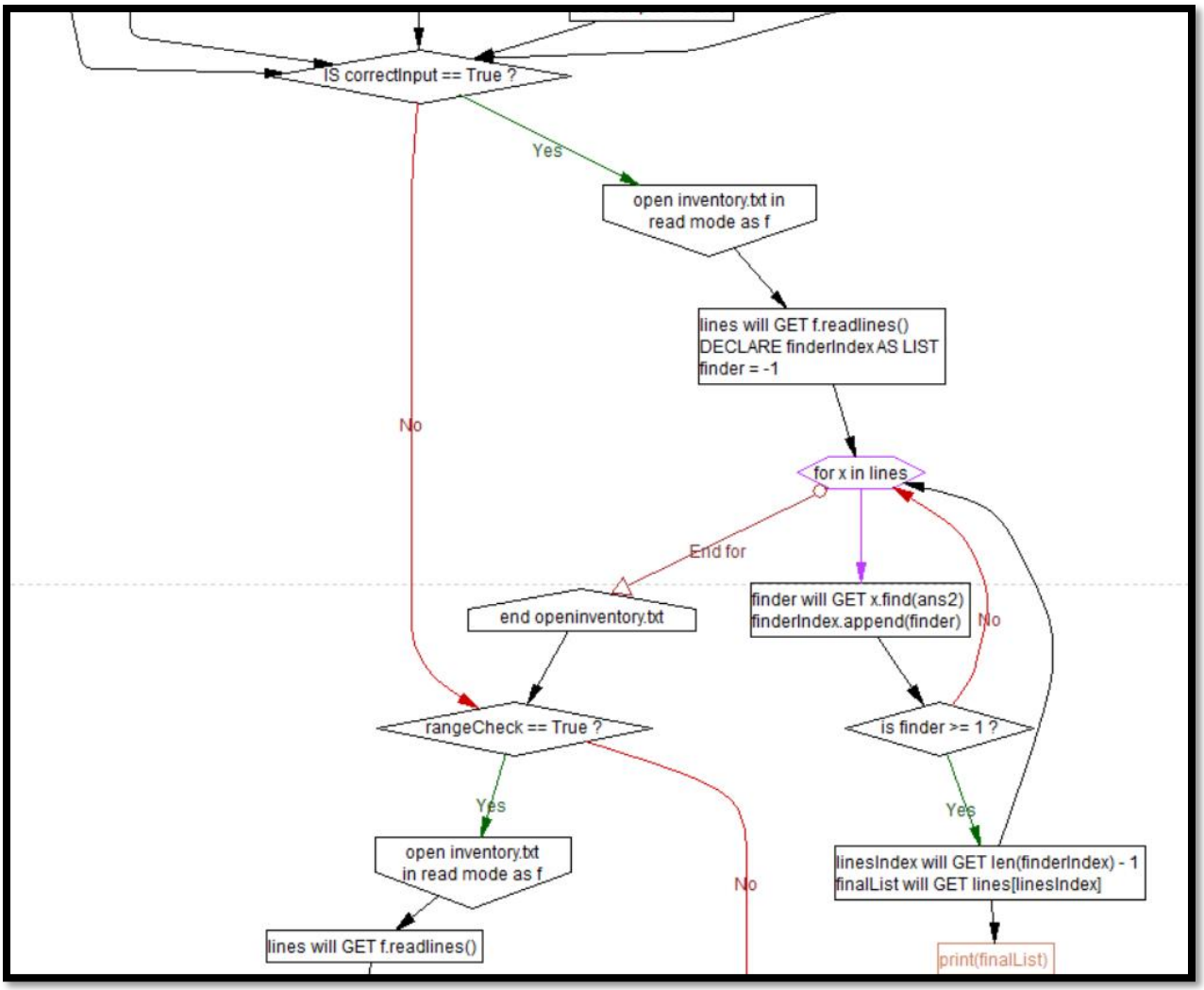
Pseudocode Stock Replenish Function:

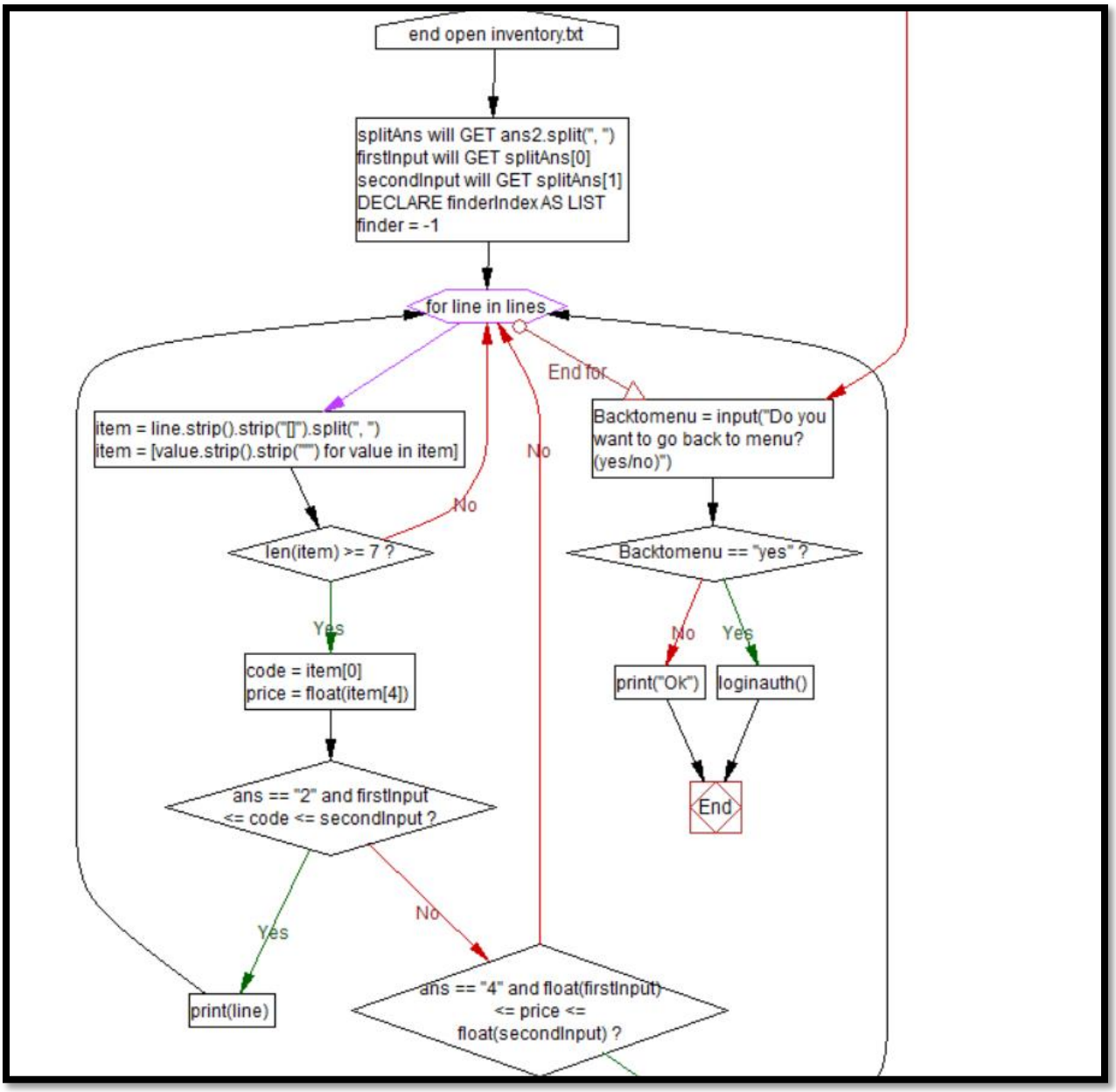




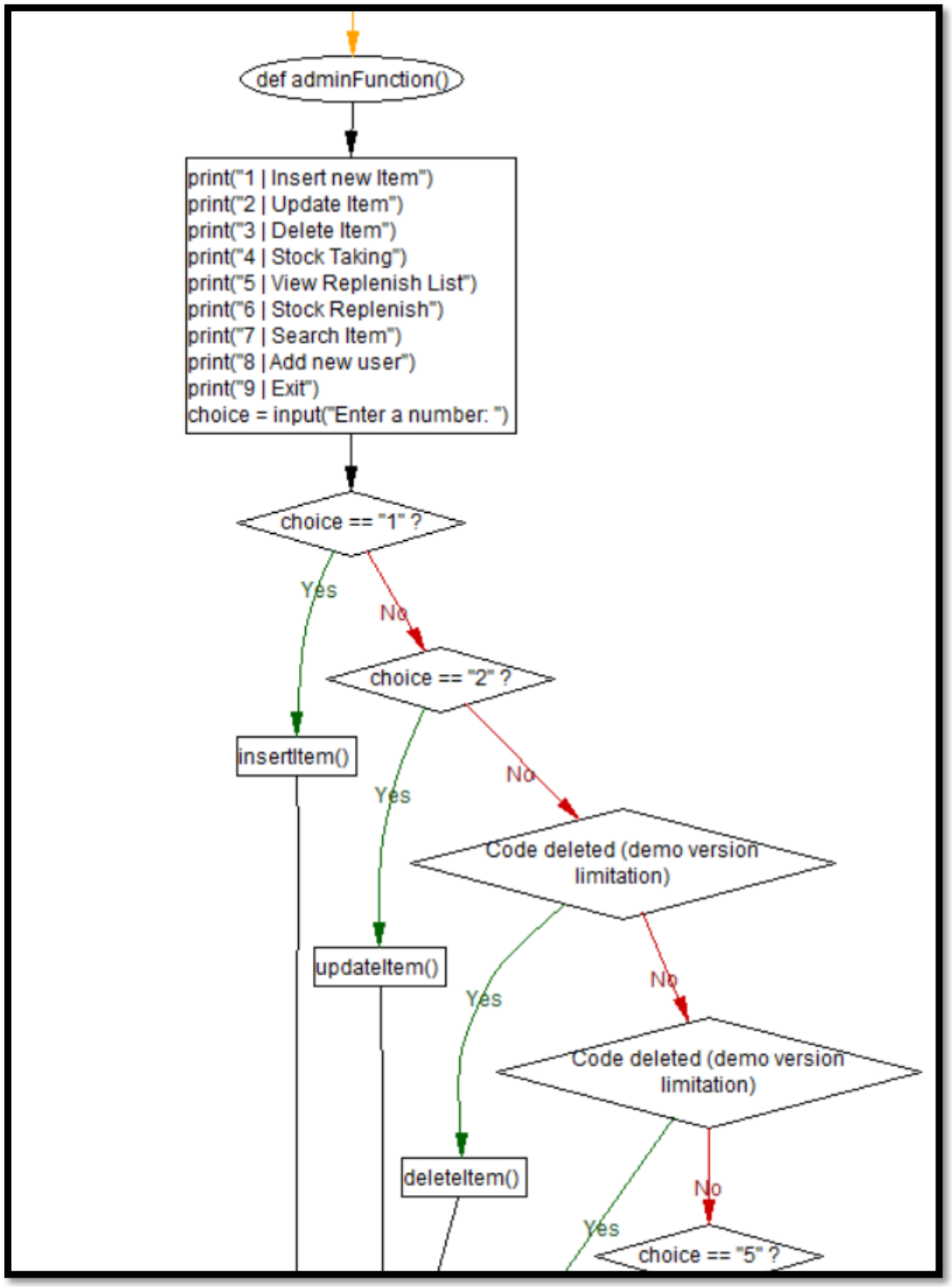
Pseudocode Search Item Function:

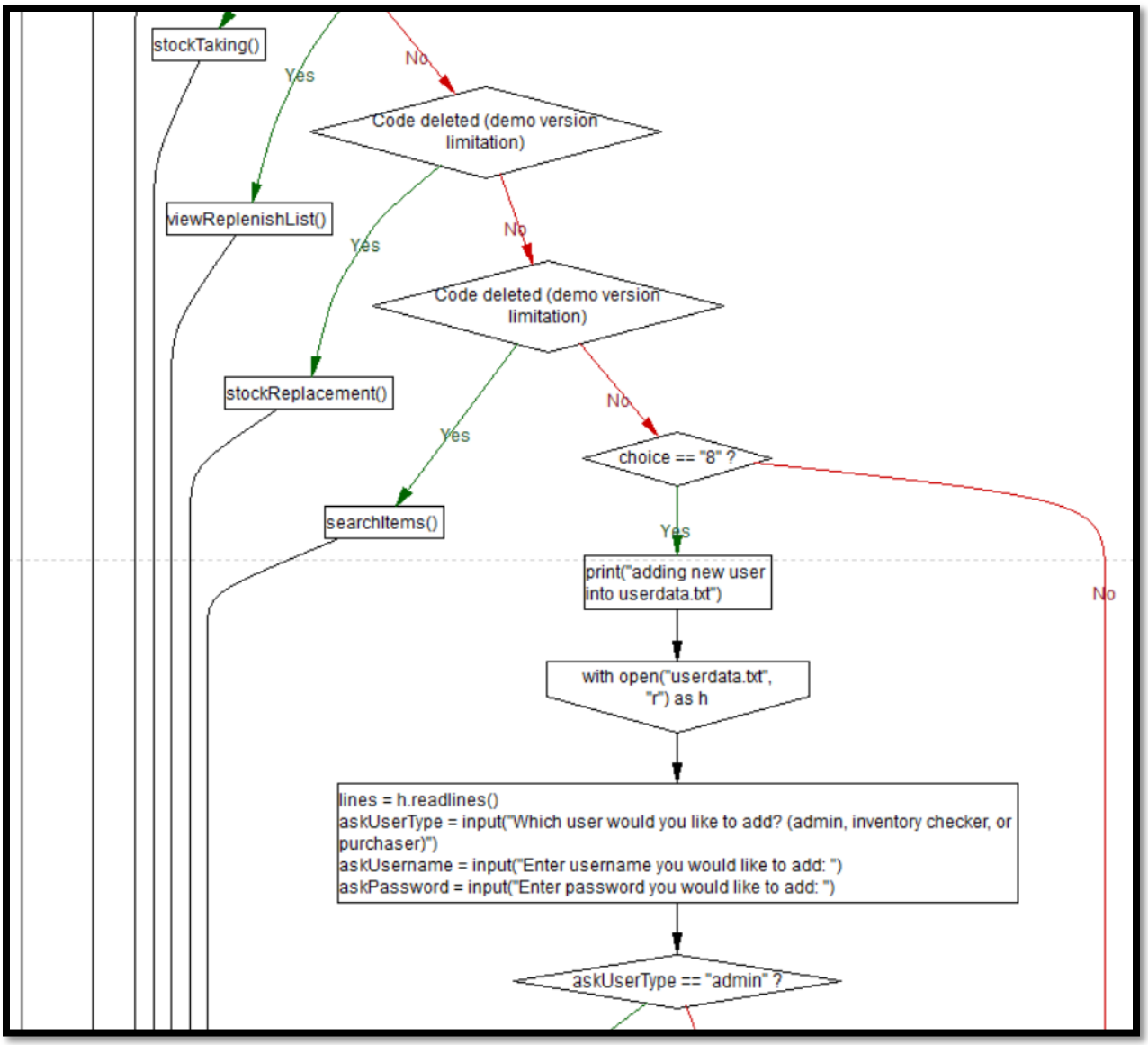

searchItems()
print("1 | Search item by description: " "\n" "2 | Search item by code range.")
print("3 | Search items in a specific category")
print("4 | Search items in a specific price range.")
ans = input("Enter a number from 1 to 4: ")
correctInput will GET True
rangeCheck will GET False
IS ans == "1" ?
Yes
No
is ans == "2"?
Yes
No
ans2 = input("Enter description: ")
ans2 = input("Enter a code range (ex. 3000, 5000): ")
is ans == "3" ?
rangeCheck will GET True
correctInput will GET False
No
Yes
is ans == "4" ?
Yes
No
ans2 = input("Enter a price range (ex. 30, 50): ")
ans2 = input("Enter a category: ")
rangeCheck will GET True
correctInput will GET False
print("Your entry was out of the range")
correctInput = False

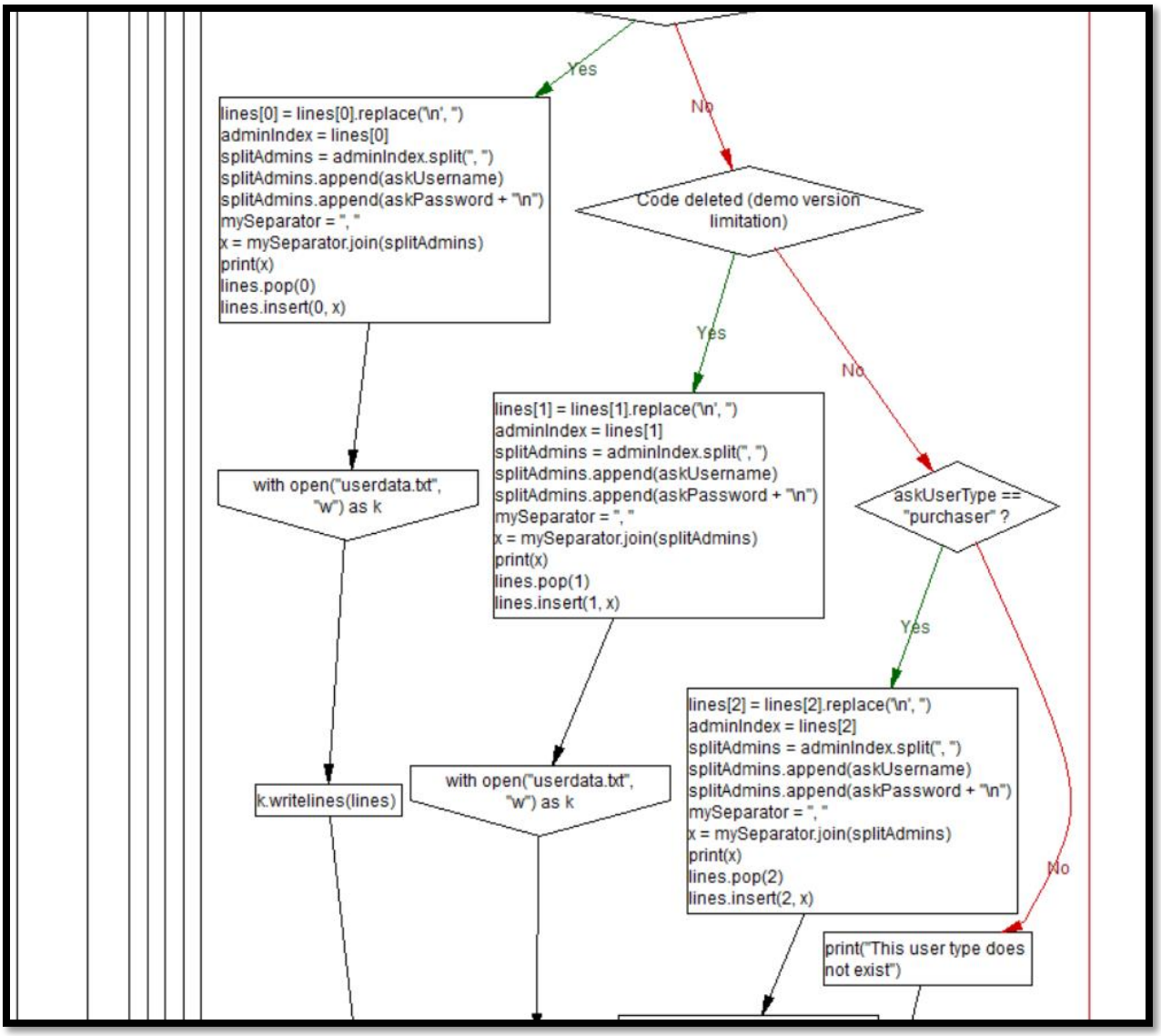

IS correctInput == True ?
Yes
No
open inventory.txt in
read mode as f
lines will GET f.readlines()
DECLARE finderIndex AS LIST
finder = -1
for x in lines
End for
end openinventory.txt
finder will GET x.find(ans2)
finderIndex.append(finder)
No
rangeCheck == True ?
is finder >= 1 ?
Yes
Yes
open inventory.txt
in read mode as f
No
linesIndex will GET len(finderIndex) - 1
finalList will GET lines[linesIndex]
lines will GET f.readlines()
print(finalList)

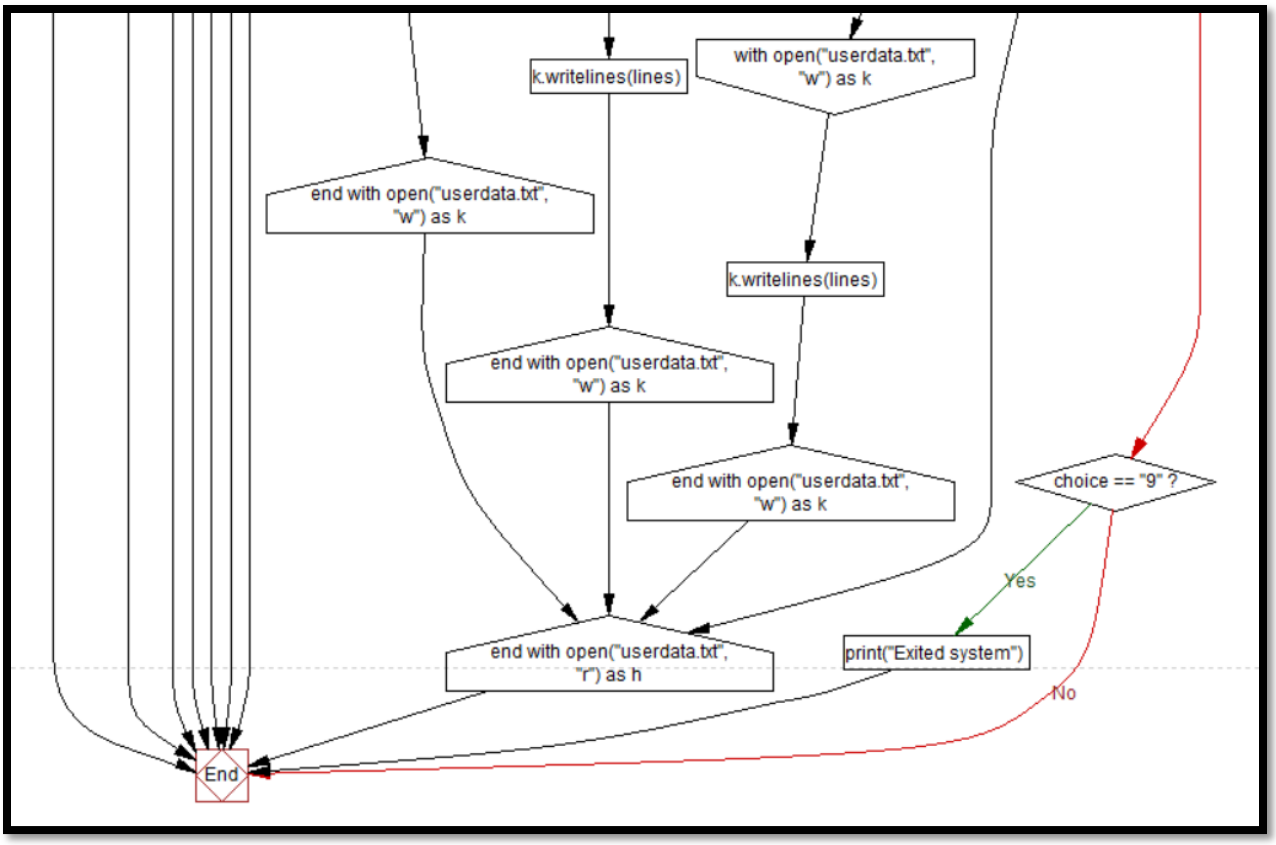

end open inventory.txt
splitAns will GET ans2.split(", ")
firstInput will GET splitAns[0]
secondInput will GET splitAns[1]
DECLARE finderIndex AS LIST
finder = -1
for line in lines
End for
item = line.strip().strip("[]").split(", ")
item = [value.strip().strip("'") for value in item]
No
len(item) >= 7 ?
Yes
code = item[0]
price = float(item[4])
ans == "2" and firstInput
<= code <= secondInput ?
Yes
No
print(line)
ans == "4" and float(firstInput)
<= price <=
float(secondInput) ?
No
Backtomenu = input("Do you
want to go back to menu?
(yes/no)")
Backtomenu == "yes" ?
No
Yes
print("Ok")
loginauth()
End

Flowchart Admin Function:

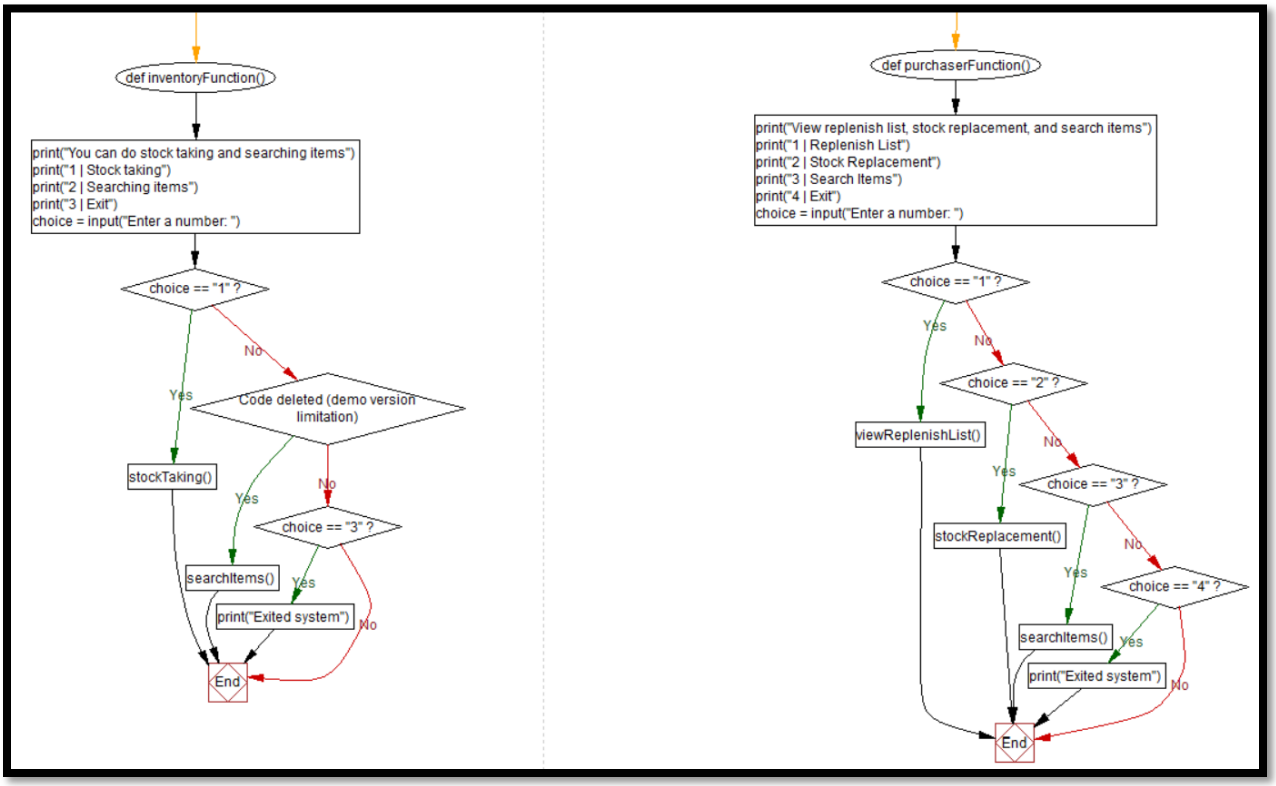



stockTaking()
Yes
No
Code deleted (demo version limitation)
viewReplenishList()
Yes
No
Code deleted (demo version limitation)
stockReplacement()
Yes
No
choice == "8" ?
searchItems()
Yes
No
print("adding new user into userdata.txt")
with open("userdata.txt", "r") as h
lines = h.readlines()
askUserType = input("Which user would you like to add? (admin, inventory checker, or purchaser)")
askUsername = input("Enter username you would like to add: ")
askPassword = input("Enter password you would like to add: ")
askUserType == "admin" ?


Yes
No
lines[0] = lines[0].replace('\n', '')
adminIndex = lines[0]
splitAdmins = adminIndex.split(", ")
splitAdmins.append(askUsername)
splitAdmins.append(askPassword + "\n")
mySeparator = ", "
x = mySeparator.join(splitAdmins)
print(x)
lines.pop(0)
lines.insert(0, x)
Code deleted (demo version limitation)
Yes
No
lines[1] = lines[1].replace('\n', '')
adminIndex = lines[1]
splitAdmins = adminIndex.split(", ")
splitAdmins.append(askUsername)
splitAdmins.append(askPassword + "\n")
mySeparator = ", "
x = mySeparator.join(splitAdmins)
print(x)
lines.pop(1)
lines.insert(1, x)
with open("userdata.txt", "w") as k
askUserType == "purchaser" ?
Yes
No
lines[2] = lines[2].replace('\n', '')
adminIndex = lines[2]
splitAdmins = adminIndex.split(", ")
splitAdmins.append(askUsername)
splitAdmins.append(askPassword + "\n")
mySeparator = ", "
x = mySeparator.join(splitAdmins)
print(x)
lines.pop(2)
lines.insert(2, x)
with open("userdata.txt", "w") as k
k.writelines(lines)
print("This user type does not exist")



Flowchart Inventory and Purchaser Function:



Code Theory and explanation:

```
def insertItem():
    newItem = []
    alreadyExists = False
    while alreadyExists == False: #While loop exists to ensure code chosen doesn't
already exist
        codeAdd = input("Enter code of your item: ")
        with open("inventory.txt", 'r') as f:
            lines = f.readlines()
        finderIndex = []
        finder = -1 #find function returns -1 at negative output and 2 at positive output
        used = False
        for x in lines:
            finder = x.find(codeAdd)
            finderIndex.append(finder)
            if finder >= 1: #output of 2 by find function implies the code has been found
in the list
                alreadyExists = False #makes entire while loop repeat
                used = True #prevents code from progressing to next input
                print("This code is already being used, pick another one.")
        if used == False:
           alreadyExists = True
    descriptionAdd = input("Enter description of your item: ")
    categoryAdd = input("Enter category of your item: ")
    unitAdd = input("Enter unit of your item: ")
    priceAdd = input("Enter price of your item: ")
    quantityAdd = input("Enter quantity of your items: ")
    minimumAdd = input("Enter minimum number of items allowed of this: ")
    newItem.append(codeAdd)
    newItem.append(descriptionAdd)
    newItem.append(categoryAdd)
    newItem.append(unitAdd)
    newItem.append(priceAdd)
    newItem.append(quantityAdd)
    newItem.append(minimumAdd)
    finalList = str(newItem)
    with open('inventory.txt', 'a') as f:
        f.write('\n')
        f.write(finalList)
    print("Item has been added to the inventory.")
    Backtomenu = input("Do you want to go back to menu? (yes/no)")
    if Backtomenu == "yes":
        loginauth()
    else:
        print("Ok")
```

Utilizing the method insertItem() from the example code, the user may add a new item to the inventory. Let's look at the code and explain how it functions:

The function starts by initializing the list newItem with an empty value and setting the boolean variable alreadyExists' value to False.

The while loop is initiated by the function when the alreadyExists == False condition is satisfied. Before inputting the code for the new item, this loop makes sure it doesn't already exist in the inventory.

The user must supply the object's code in order for it to be included in the loop. The code opens the "inventory.txt" file in read-only mode before reading the data into the lines variable.

The outcomes of the find() method for each line in the file are kept in a list called finderIndex. The default value of the variable finder is 1.

Use the find() method to determine if the enter code exists anywhere in the questioned lines. The alreadyExists variable is set to False if the code is discovered (i.e., finder >= 1), denoting that the code is already present in the inventory. To cease processing the incoming data, the variable is updated to True.

If the code cannot be located, the while loop will end (used == False), demonstrating that the alreadyExists variable is set to True and the code is original.

The user is prompted to input the new item's category, price, quantity, and minimum number of units in a popup window. These inputs are kept in the newItem list.

The newItem list is converted to a string using the str() method, which is then stored in the finalList variable.

The "inventory.txt" file is opened in append mode and the finalList string, which contains the details of the new item, is written there. The file is closed after writing.

A notification telling the user that the item has been successfully added to the inventory is printed.

The choice of "yes" or "no" will return the user to the menu. When "yes" is entered, the loginauth() function is called, taking the user back to the main menu. If not, the function is terminated and "Ok" is written instead.

```
def updateItem():
    ask = input("Enter product code of item you would like to update: ")
    with open("inventory.txt", 'r') as f:
        lines = f.readlines()
    finderIndex = []
    finder = -1
    for x in lines:
        finder = x.find(ask)
        finderIndex.append(finder)
        if finder >= 1:
            linesIndex = len(finderIndex) - 1
            finalList = lines[linesIndex]
            print(finalList)
            question = input("What would you like to change? ")
            ans = input("What would you like to replace it with? ")
            changedList = finalList.replace(question, ans)
            print(changedList)
            lines.pop(linesIndex)
            lines.insert(linesIndex, changedList)
            print(lines)
            with open("inventory.txt", 'w') as file:
                file.writelines(lines)
    Backtomenu = input("Do you want to go back to menu? (yes/no)")
    if Backtomenu == "yes":
        loginauth()
    else:
        print("Ok")
```

At the beginning of the method, the user must enter the product code for the item they wish to change. The input is stored in the ask variable.

When the code reads the "inventory.txt" file in read mode, the lines variable is populated with the data it has gleaned.

Both Finder and finderIndex are created as empty lists. Finder has a beginning value of 1, while finderIndex will store the results of the search() method.

The find() method is used to do a recursive search across each line in each line using the user-inputted product code. The value returned by find() is added to the enlarged FinderIndex.

If the product code is found (i.e., finder >= 1), the code keeps updating the item.

The finderIndex index of the final product code instance, which corresponds to the lines list index of the item's information, serves as the value of the variable linesIndex.

The item's details are extracted from the lines using lines[linesIndex] and saved in the finalList variable.

The user is asked to make a modification, and the desired alteration is recorded in the inquiry variable.

The user is asked how much the item would cost to replace, and their response is kept in the ans variable.

The replace() function constructs the changedList variable by replacing the supplied value with the value given in ans inside the finalList string.

To print the information of the modified item, use the changedList.

Using lines.pop(linesIndex), the first line of the item is deleted from the lines list.

At the same index using lines, the updated changedList is added to the lines list.It should be (lineIndex, changedList).

The "inventory.txt" file inserts a fresh lines list in place of the prior data. The syntax open("inventory.txt", "w") opens the file in write mode.

The user will be taken back to the menu after selecting "yes" or "no". Inputting "yes" causes the loginauth() method to be called, which takes the user back to the main menu. If not, "Ok" is typed instead and the function is ended.

```
ef deleteItem():
        ask = input("Enter Product Code of item you would like deleted: ")
        with open("inventory.txt", 'r') as f:
             lines = f.readlines()
        finderIndex = []
        finder = -1
        for x in lines:
             finder = x.find(ask)
             finderIndex.append(finder)
             if finder >= 1:
                  linesIndex = len(finderIndex) - 1
                  finalList = lines[linesIndex]
                  print("This item has been deleted: " + finalList)
                  lines.pop(linesIndex)
                  lines.pop(linesIndex)
                  print(lines)
                  with open("inventory.txt", 'w') as file:
                    file.writelines(lines)
        Backtomenu = input("Do you want to go back to menu? (yes/no)")
        if Backtomenu == "yes":
            loginauth()
        else:
            print("Ok")
```

The user can take an item out of the inventory by calling the method deleteItem(), which is detailed in the code that is given. The following statement explains how the code works:

To utilize the feature, the user must first provide the product code of the item they wish to delete. The ask variable holds the input.

The lines variable is filled with the information the code has gathered after reading the "inventory.txt" file in read mode.

When constructed, Finder and finderIndex are both empty lists. The starting value of Finder is 1, whereas finderIndex stores the outcomes of the search() function.

Using the user-inputted product code, the find() function does a recursive search over each line in each line. The FinderIndex is expanded by the value returned by find().

If the product code is found (i.e., finder >= 1), the procedure continue removing the item.

The finderIndex index of the final product code instance, which corresponds to the lines list index of the item's information, serves as the value of the variable linesIndex.

The item's details are extracted from the lines using lines[linesIndex] and saved in the finalList variable.

A message stating that the item has been removed is written along with the contents of finalList.

Using lines.pop(linesIndex), the item's line is twice removed from the list of lines. This removes the line holding the item's information as well as the following line break.

The "inventory.txt" file's prior text is replaced with the list of changed lines. The syntax open("inventory.txt", "w") opens the file in write mode.

The user must select "yes" or "no" in order to get back to the menu. The loginauth() method is launched when "yes" is entered, bringing the user back to the main menu. If not, the program writes "Ok" and ends the function.

```
def stockTaking():
    ask = input("Enter Product Code of item whose stock you would like updated: ")
    with open("inventory.txt", 'r') as f:
        lines = f.readlines()
    finderIndex = []
    finder = -1
    for x in lines:
        finder = x.find(ask)
        finderIndex.append(finder)
        if finder >= 1:
            linesIndex = len(finderIndex) - 1
            finalList = lines[linesIndex]
            print(finalList)
            splitlist = finalList.split(',')
            print("Quantity of stock is " + splitlist[5])
            ans = input("Would you like to change this quantity(yes/no)? ")
            if ans == "yes":
                ask2 = input("What would you like to change it to? ")
                splitlist.pop(5)
                splitlist.insert(5, " '" + ask2 + "'")
                mySeparator = ","
                y = mySeparator.join(splitlist)
                print(y)
                lines.pop(linesIndex)
                lines.insert(linesIndex, y)
                with open("inventory.txt", 'w') as file:
                    file.writelines(lines)

            elif ans == "no":
                print("Okay")
            else:
                print("That is not a valid input: ")
    Backtomenu = input("Do you want to go back to menu? (yes/no)")
    if Backtomenu == "yes":
        loginauth()
    else:
        print("Ok")
```

The user may change the stock quantity of an item in the inventory by using the stockTaking() function, which is specified in the code that is given. The following sentence explains how the code works:

The function asks the user for the product code of the item whose stock level they want to change as its first step. The ask variable holds the input.

The lines variable is filled with the information obtained from the "inventory.txt" file when the code reads it in read mode.

Finder, which has a starting value of 1, and finderIndex, which will hold the outcomes of the search() function, are both constructed as empty lists.

The user-inputted product code is searched for using the find() function recursively over each line in each line. FinderIndex is expanded with the value returned by find().

The code then maintains updating the stock quantity if the product code is discovered (i.e., finder >= 1).

The value of the variable linesIndex is the finderIndex index of the last instance of the product code, which corresponds to the lines list index of the item's information.

Using lines[linesIndex], the information about the item is taken from the lines and saved in the finalList variable.

A comma is used as the separator to separate the finalList into a list named splitlist.

The list of lines that use lines at the same index is expanded to include the modified line y.(linesIndex, y) is inserted.

The "inventory.txt" file's prior text is replaced with the list of changed lines. The syntax open("inventory.txt", "w") opens the file in write mode.

The user must select "yes" or "no" in order to get back to the menu. The loginauth() method is launched when "yes" is entered, bringing the user back to the main menu. If not, the program writes "Ok" and ends the function.

```
def viewReplenishList():
    with open("inventory.txt", 'r') as f:
        lines = f.readlines()
    print("Replenish List:")
    print("~~~~~~~~~~~~~~~~~~~")
    for x in lines:
        item = x.strip().strip("[]").split(",")
        item = [value.strip().strip("'") for value in item]
        if len(item) >= 7:
            code = item[0]
            description = item[1]
            category = item[2]
            unit = item[3]
            price = item[4]
            quantity = int(item[5])
            minimum = int(item[6])
            if quantity < minimum:
                print("Code: {}".format(code))
                print("Description: {}".format(description))
                print("Category: {}".format(category))
                print("Unit: {}".format(unit))
                print("Price: {}".format(price))
                print("Quantity: {}".format(quantity))
                print("Minimum: {}".format(minimum))
                print("~~~~~~~~~~~~~~~~~~~")
                Backtomenu = input("Do you want to go back to menu? (yes/no)")
                if Backtomenu == "yes":
                    loginauth()
                else:
                    print("Ok")
```

The function reads information into the lines variable from the "inventory.txt" file using the readlines() method.

"Replenish List:" is the header of the refill list.

As the code cycles over each line in a line, it processes the data.

Each line is stripped of any previous or after whitespace in order to construct an item list, and square brackets are used as the line separator before a comma.

Remove all single quotes and spaces from each value in the item using list comprehension.

If the item's length is higher than or equal to 7, which indicates that all required item data is accessible, the item is processed further.

Extractions are made of the item's code, description, category, unit, price, quantity, and minimum information.

The quantity and minimum values are converted to integers using the int() method.

If the amount falls below the minimum, the item details are displayed, indicating the need for replacement.

The user is requested to select "yes" or "no" to return to the menu for each item on the refill list. Inputting "yes" causes the loginauth() function to run, returning the user to the main menu. If not, "Ok" is written and the application terminates the function.

```
def stockReplenish():
    ask = input("Enter code of item you want replenished: ")
    with open("inventory.txt", 'r') as f:
        lines = f.readlines()
        finderIndex = []
        finder = -1
        for x in lines:
            finder = x.find(ask)
            finderIndex.append(finder)
            if finder >= 1:
                linesIndex = len(finderIndex) - 1
                finalList = lines[linesIndex]
                splitList = finalList.strip().strip("[]").split(",")
                splitList = [y.strip().strip("'") for y in splitList]
                print(splitList)
                print("Quantity is: " + splitList[5])
                ask2 = input("Would you like to update the quantity? (yes/no) ")
                if ask2 == "yes":
                    ask3 = input("Enter new quantity: ")
                    print(splitList[5])
                    replacedList = finalList.replace(splitList[5], ask3)
                    print(replacedList)
                    lines.pop(linesIndex)
                    lines.insert(linesIndex, replacedList)
                    with open("inventory.txt", 'w') as file:
                        file.writelines(lines)
    Backtomenu = input("Do you want to go back to menu? (yes/no)")
    if Backtomenu == "yes":
        loginauth()
    else:
        print("Ok")
```

The ask variable stores the user's response once the user is prompted to enter the item's code.

The readlines() method reads the contents of the "inventory.txt" file into the lines variable after opening it in read mode.

The item with the matching code is found by iterating over each line in a loop. The code is searched using the find() method, and the results are saved in the finder variable.

The user must select "yes" or "no" in order to get back to the menu. The loginauth() method is launched when "yes" is entered, bringing the user back to the main menu. If not, the program writes "Ok" and ends the function.

```
def searchItems():
    print("1 | Search item by description: " "\n" "2 | Search item by code range." )
    print("3 | Search items in a specific category")
    print("4 | Search items in a specific price range.")

    ans = input("Enter a number from 1 to 4: ")
    correctInput = True
    rangeCheck = False
    if ans == "1":
```

```
        ans2 = input("Enter description: ")
    elif ans == "2":
        ans2 = input("Enter a code range (ex. 3000, 5000): ")
        rangeCheck = True
        correctInput = False
    elif ans == "3":
        ans2 = input("Enter a category: ")
    elif ans == "4":
        ans2 = input("Enter a price range (ex. 30, 50): ")
        rangeCheck = True
        correctInput = False
    else:
        print("Your entry was out of the range")
        correctInput = False

    if correctInput == True:
        with open("inventory.txt", 'r') as f:
            lines = f.readlines()
            finderIndex = []
            finder = -1
            for x in lines:
                finder = x.find(ans2)
                finderIndex.append(finder)
                if finder >= 1:
                    linesIndex = len(finderIndex) - 1
                    finalList = lines[linesIndex]
                    print(finalList)
    if rangeCheck == True:
        with open("inventory.txt", 'r') as f:
            lines = f.readlines()
        splitAns = ans2.split(", ")
        firstInput = splitAns[0]
        secondInput = splitAns[1]
        finderIndex = []
        finder = -1
        for line in lines:
            item = line.strip().strip("[]").split(", ")
            item = [value.strip().strip("'") for value in item]
            if len(item) >= 7:
                code = item[0]
                price = float(item[4])
                if ans == "2" and firstInput <= code <= secondInput:
                    print(line)
                elif ans == "4" and float(firstInput) <= price <= float(secondInput):
                    print(line)
    Backtomenu = input("Do you want to go back to menu? (yes/no)")
    if Backtomenu == "yes":
        loginauth()
    else:
        print("Ok")
```

As soon as the function is activated, a menu with four search options—description, code range, category, and price range—appears.

To select a search option, the user must provide a number between 1 and 4. The ans variable holds the input.

Before deciding on the preferred search option, the code first validates the value of ans, and it then executes appropriately.

The user is prompted to enter a description if they select option 1. The ans2 variable holds the input.

```
def adminFunction():
    print("1 | Insert new Item")
    print("2 | Update Item")
    print("3 | Delete Item")
```

```
    print("4 | Stock Taking")
    print("5 | View Replenish List")
    print("6 | Stock Replenish")
    print("7 | Search Item")
    print("8 | Add new user")
    print("9 | Exit")
    choice = input("Enter a number: ")
    if choice == "1":
        insertItem()
    elif choice == "2":
        updateItem()
    elif choice == "3":
        deleteItem()
    elif choice == "4":
        stockTaking()
    elif choice == "5":
        viewReplenishList()
    elif choice == "6":
        stockReplenish()
    elif choice == "7":
        searchItems()
    elif choice == "8":
        print("adding new user into userdata.txt")
        with open("userdata.txt", "r") as h:
            lines = h.readlines()
            askUserType = input("Which user would you like to add? (admin, inventory
checker, or purchaser)")
            askUsername = input("Enter username you would like to add: ")
            askPassword = input("Enter password you would like to add: ")
            if askUserType == "admin":
                lines[0] = lines[0].replace('\n', '')
                adminIndex = lines[0]
                splitAdmins = adminIndex.split(", ")
                splitAdmins.append(askUsername)
                splitAdmins.append(askPassword + "\n")
                mySeparator = ", "
                x = mySeparator.join(splitAdmins)
                print(x)
                lines.pop(0)
                lines.insert(0, x)
                with open("userdata.txt", "w") as k:
                    k.writelines(lines)
            elif askUserType == "inventory checker":
                lines[1] = lines[1].replace('\n', '')
                adminIndex = lines[1]
                splitAdmins = adminIndex.split(", ")
                splitAdmins.append(askUsername)
                splitAdmins.append(askPassword + "\n")
                mySeparator = ", "
                x = mySeparator.join(splitAdmins)
                print(x)
                lines.pop(1)
                lines.insert(1, x)
                with open("userdata.txt", "w") as k:
                    k.writelines(lines)
            elif askUserType == "purchaser":
                lines[2] = lines[2].replace('\n', '')
                adminIndex = lines[2]
                splitAdmins = adminIndex.split(", ")
                splitAdmins.append(askUsername)
                splitAdmins.append(askPassword + "\n")
                mySeparator = ", "
                x = mySeparator.join(splitAdmins)
```

```
                print(x)
                lines.pop(2)
                lines.insert(2, x)
                with open("userdata.txt", "w") as k:
                    k.writelines(lines)
            else:
                print("This user type does not exist")

    elif choice == "9":
        print("Exited system")
```

The function shows a menu with a variety of choices, such as adding a new item, editing an item, removing an item, taking an inventory, seeing the list of goods that need to be refilled, replenishing stock, looking for an item, adding a new user, and leaving the application.

Before selecting a choice, the user must enter a number. The chosen variable contains the input.

The function calls the appropriate function to perform the desired action based on the supplied option. For instance, if the insertItem() method is used, the option is "1".

If "8" is chosen, a new user can be added by providing the username, password, and user type (administrator, inventory checker, or purchaser). After reading the current user data from "userdata.txt" and "userdata.txt," it modifies the corresponding user type with the new username and password. The modified user data is then added to the file.

If the user chooses option "9", the function prints "Exited system" to indicate that the program has terminated.

```
def inventoryFunction():
    print("1 | Stock taking")
    print("2 | Search item")
    print("3 | Exit")
    choice = input("Enter a number: ")
    if choice == "1":
        stockTaking()
    elif choice == "2":
        searchItems()
    elif choice == "3":
        print("Exited system")
```

A menu with options for taking a stock, looking for a specific item, and quitting the software is displayed by the function.

In order to select an option, the user must provide a number. The chosen variable contains the input.

The function calls the appropriate function to perform the desired action based on the selected option. For instance, if choice is set to "1", the stockTaking() method is called.

The searchItems() method is used if the option "2" is used.

Option "3" causes the function to output "Exited system" once the application terminates if the user selects that option.

```
def purchaserFunction():
    print("1 | Replenish List")
    print("2 | Stock Replenish")
    print("3 | Search Item")
    print("4 | Exit")
    choice = input("Enter a number: ")
    if choice == "1":
        viewReplenishList()
    elif choice == "2":
        stockReplenish()
    elif choice == "3":
        searchItems()
```

```
    elif choice == "4":
        print("Exited system")
```

The feature shows a menu with options for seeing the list of stock replenishments, looking for a specific item, and exiting the application.

In order to select an option, the user must provide a number. The chosen variable contains the input.

The function calls the appropriate function to perform the desired action based on the selected option. For instance, the viewReplenishList() method is used when option is set to "1".

If option is "2," the stockReplenish() method is invoked.

The searchItems() method is called when the input value is "3".

The function outputs "Exited system" to signify that the program has terminated if the option is "4".

```
def loginauth():
    with open("userdata.txt", 'r') as g:
        admin = g.readline()
        inventory = g.readline()
        purchaser = g.readline()
    adminList = admin.split(", ")
    adminList[-1] = adminList[-1].replace("\n", "")
    inventoryList = inventory.split(", ")
    purchaserList = purchaser.split(", ")
    username = input("Enter username: ")
    password = input("Enter password: ")
    correctPass = False
    ask = input("Are you an admin, inventory checker, or a purchaser: ")
    if ask == "admin":
        for x in adminList:
                if correctPass == False:
                    while username == x:
                        index = adminList.index(x)
                        passIndex = index + 1
                        if password == adminList[passIndex]:
                            if correctPass == False:
                                print("Correct Username and Password")
                                correctPass = True
                                adminFunction()


    elif ask == "inventory checker":
        for y in inventoryList:
            if correctPass == False:
                if username == y:
                    index = inventoryList.index(y)
                    passIndex = index + 1
                    if password == inventoryList[passIndex]:
                        if correctPass == False:
                            print("Correct Username and Password")
                            correctPass = True
                            inventoryFunction()

    elif ask == "purchaser":
        for z in purchaserList:
            if correctPass == False:
                if username == z:
                    index = purchaserList.index(z)
                    passIndex = index + 1
                    if password == purchaserList[passIndex]:
                        if correctPass == False:
                            print("Correct Username and Password")
```

```
                            correctPass = True
                            purchaserFunction()


loginauth()
```

The function to read user data reads the "userdata.txt" file, which provides details on the administrator, inventory checker, and purchaser users.

The admin, inventory, and purchaser user data are retained in separate variables as lists after segmenting the file's lines.

To use the function, the user must provide their username, password, and account type (administrator, inventory checker, or purchaser).

Depending on the kind of account, the function checks to see if the supplied username and password match the pertinent user information.

If the given username and password match the admin user data, the adminFunction() function is invoked to provide admin powers.

If the specified login and password match the user information for the inventory checker, the inventoryFunction() method is invoked to provide functionality for the inventory checker.

The purchaserFunction() is invoked to offer purchaser functionality if the provided login and password match the purchaser user data.

Nothing more is done if the username and password don't match any user data or if the account type is incorrect.

I/O ScreenShots:

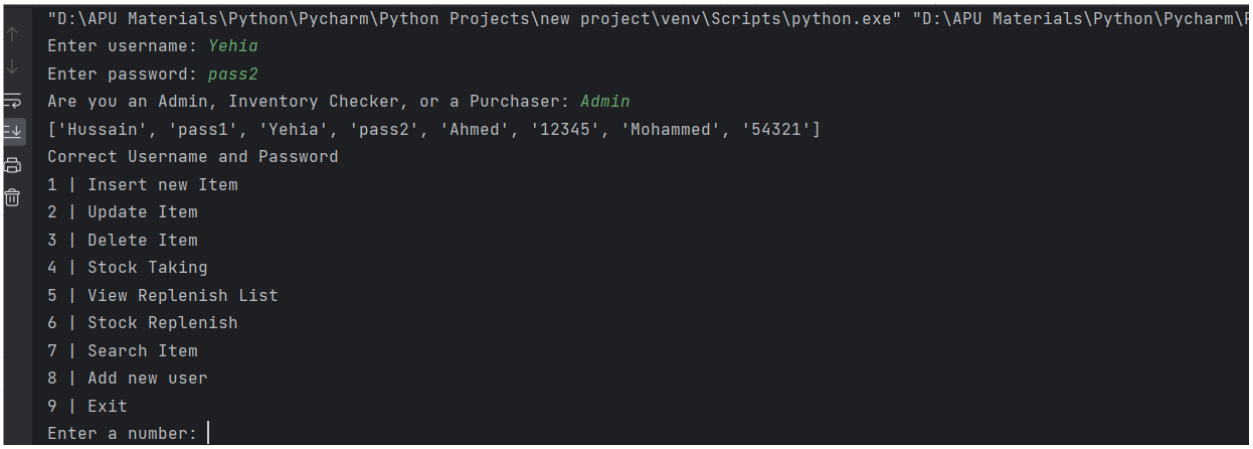
the snippet exemplifies a piece of a program that offers a login mechanism. Your login, password, and account type (admin, inventory checker, or purchaser) must be entered when requested. The application analyzes the user's access privileges based on the entered credentials and takes the necessary action.

The user submitted the following data in this case:

yehia and pass1 is an identification.

admin account type

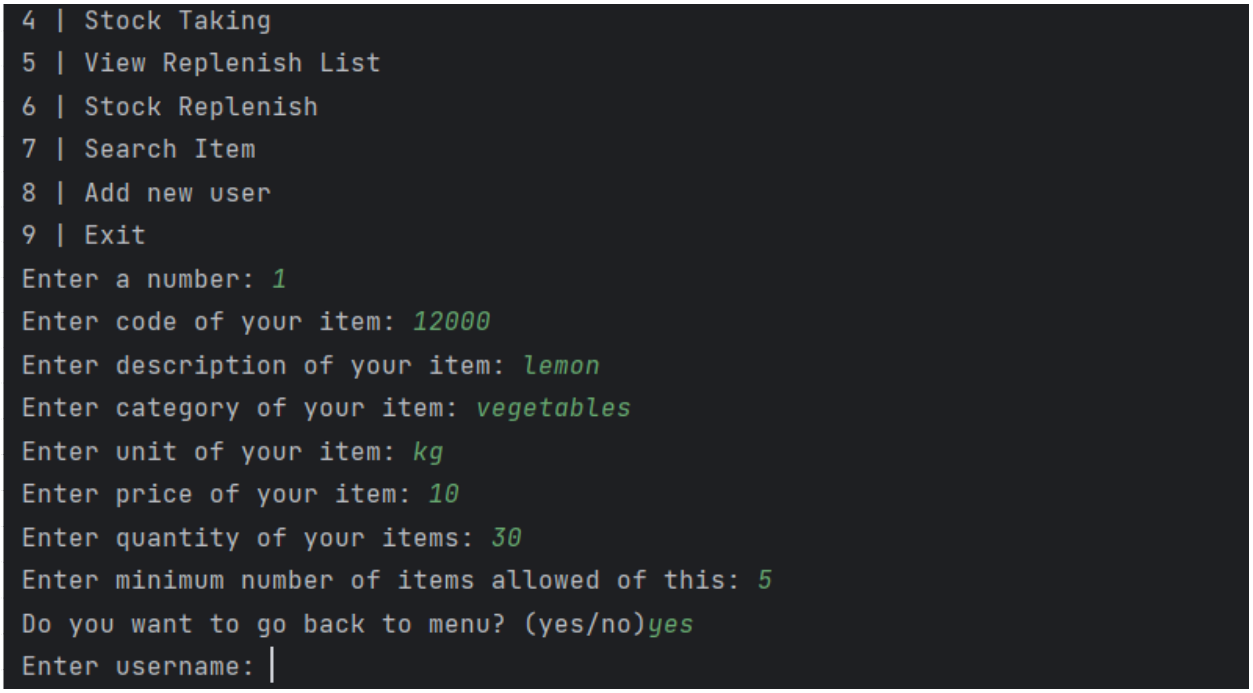
When a user enters their login information, the application compares it to the "users.txt" file's user information by using the given username, password, and account type. The software moves on to the admin portion if the provided credentials match an already-existing user account with admin access. With an exit value of 0, the application successfully finishes with no errors or exceptions.



The application performs authentication when user credentials are input to ensure their accuracy. The authentication procedure was successful since the application says that the username and password are correct.

After login in, the application displays a menu of settings for the admin user to select from. Each choice relates to a specific step in the grocery store's inventory management procedure.

The application asks the administrator to input a number after presenting the choice that goes with the required action. Typing in the relevant number will allow the administrator to select one of the choices.



A number that symbolizes the intended action under the given circumstances must be entered by the user after being presented with a choice of possibilities. The number denotes the user's decision to "Insert new Item," as seen by their selection.

The user is then prompted for additional information regarding the new item after picking the choice. The user in this instance typed in the data shown below:

Code: Lemon Veggie 12000 Describe your category

Price per kilogram: 5

30 at least five, please.

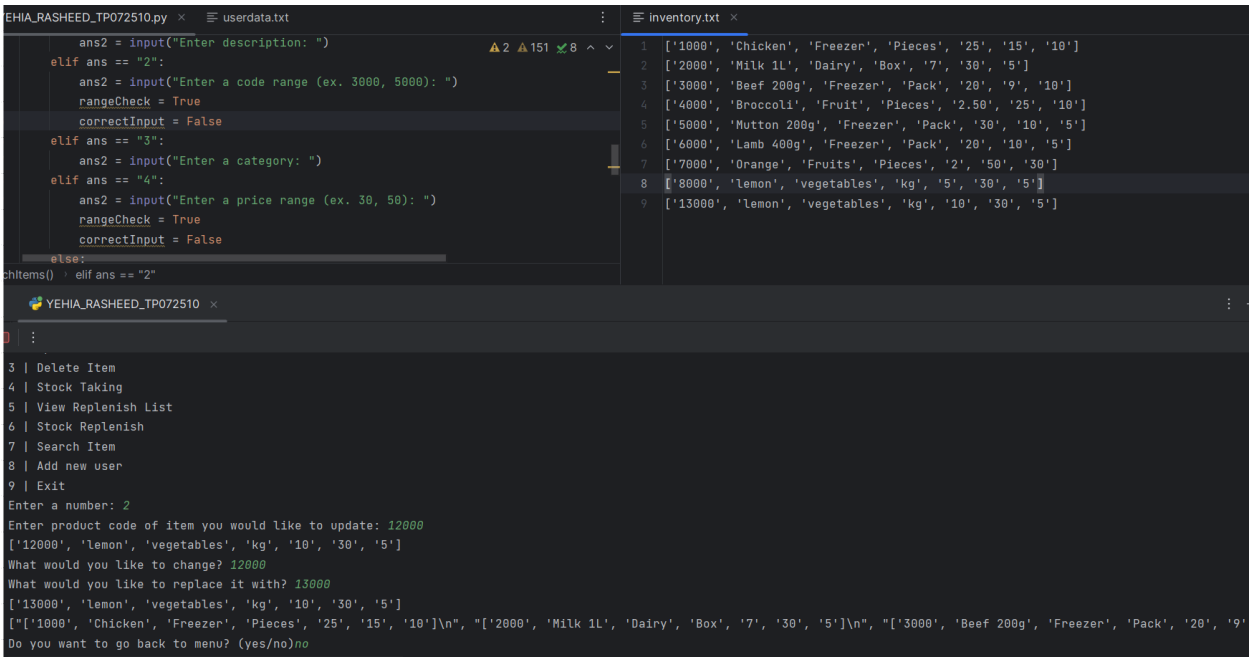
The application prompts the user to choose if they wish to return to the menu after entering the item data. The user in this instance selected "yes" in order to return to the menu.

The user is then prompted by the application to input their username.

```
Enter username: yehia
Enter password: pass1
Are you an admin, inventory checker, or a purchaser: admin
Correct Username and Password
1 | Insert new Item
2 | Update Item
3 | Delete Item
4 | Stock Taking
5 | View Replenish List
6 | Stock Replenish
7 | Search Item
8 | Add new user
9 | Exit
Enter a number: 1
Enter code of your item: 2000
This code is already being used, pick another one.
```

The software prompts the user to enter the item's code once they have made a choice. The user in this instance typed the code "2000." The application alerts the user that this code has already been used and asks them to choose another.

This notification implies that an existent item in the inventory already has the user-entered code (2000) allocated to it. The user must select an alternative code that is not currently in use in order to continue with adding a new item.

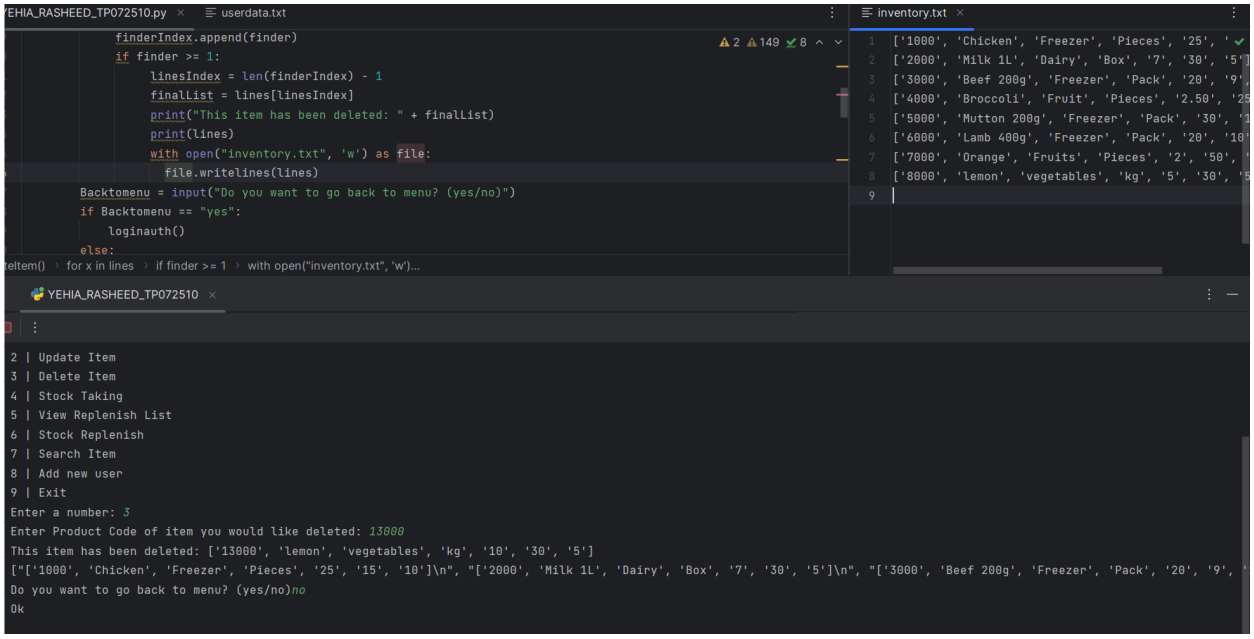


The second choice, shown by the number 2, "Update Item," was chosen by the user.

The program then displays a request for modification to the user. In this case, the user entered "12000" as the item description.

The user will then be prompted by the program to enter a new value to replace the old one. In this case, the user updated the item's code to "13000".

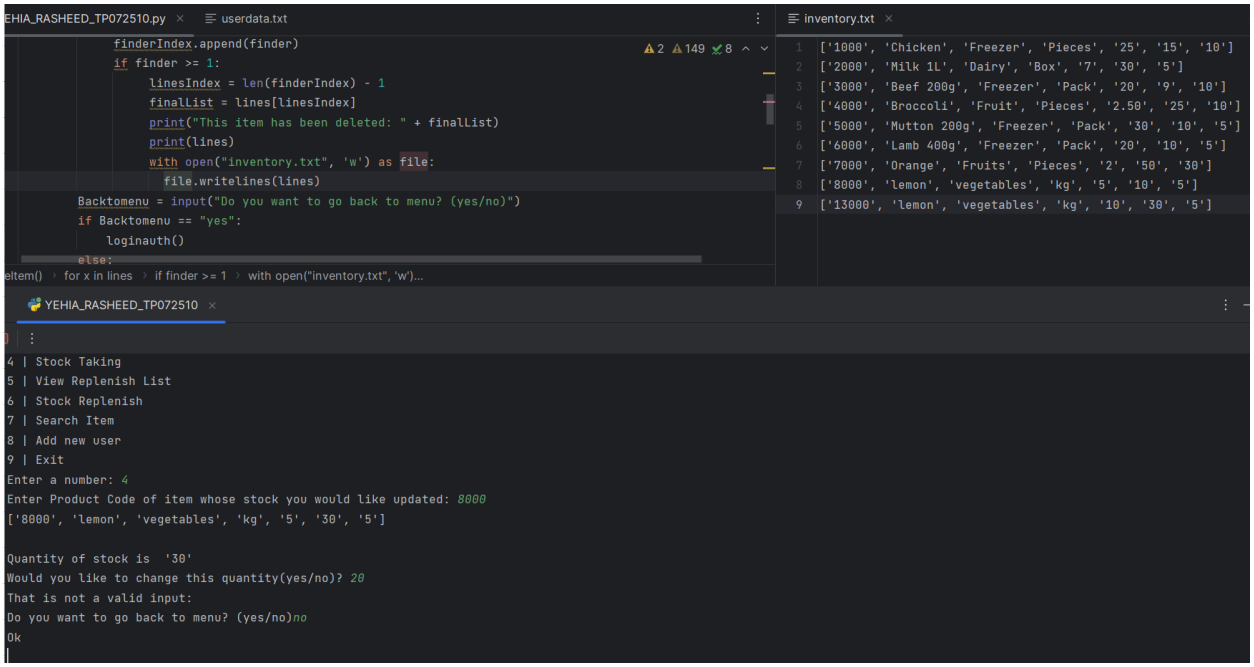
The information provided indicates that the program will probably keep getting updates. It is hard to understand the specific implementation details and effects after this without further context or code.



When a user wants to remove an item, the application asks them for the product code. The user in this instance typed "13000" as the code.

Following that, the program shows the message "This item has been deleted: ['13000', 'lemon', 'Vegetables', 'kg', '10', '30', '5']".

["['1000', 'Apples', 'Fruits', 'kg', '10', '10', '5'], "['2000', 'Bananas', 'Fruits', 'kg', '8', '15', '8']n",...] is the last line of the program that displays the products that are still in the inventory.



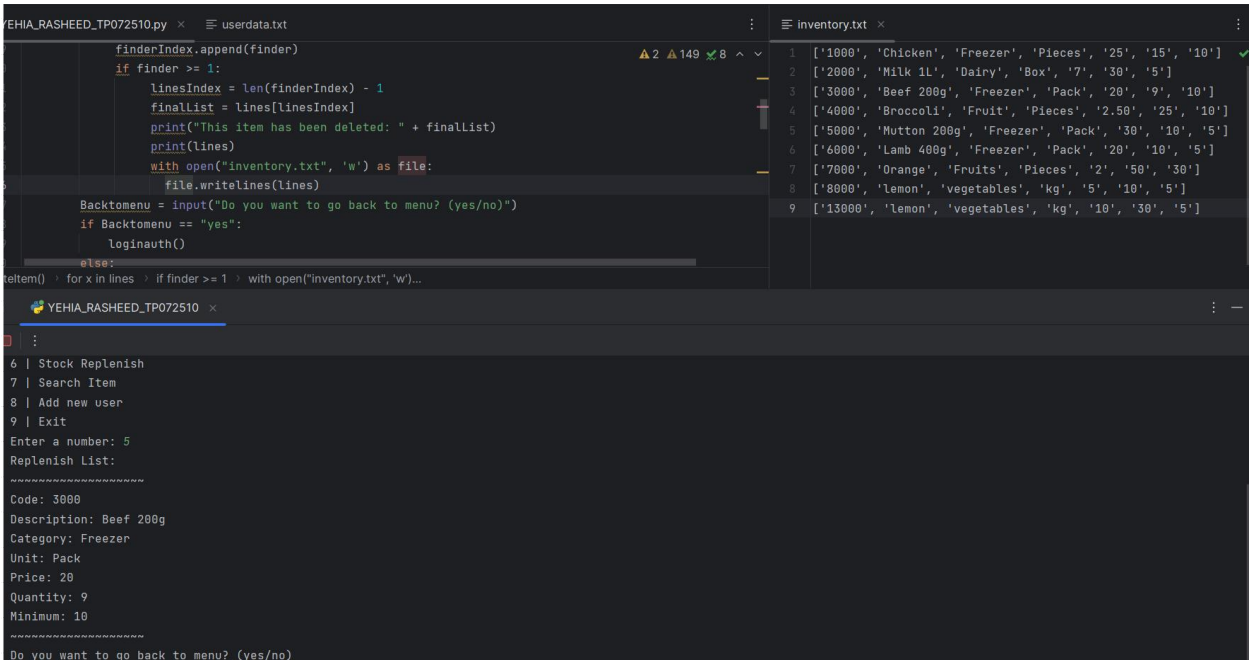
Using the provided code, the program retrieves the item from the inventory and displays its information. In this case, the item information is as follows:

(Code: 8000) Lemon Veggie Category Description

The cost per kilogram is at least $5.30 plus five.

The computer will then ask the user if they want to change the quantity of the stock item. The user responded "yes" after coming to agreement.

When the user wants to establish a new price for the item, the computer then prompts them to do so. In this case, the user entered 20



The Replenish List, which lists the inventory items that require replenishment, is shown by the application. In this instance, there is only one item in the list:

Code: 3000

information on beef

Class: freezer

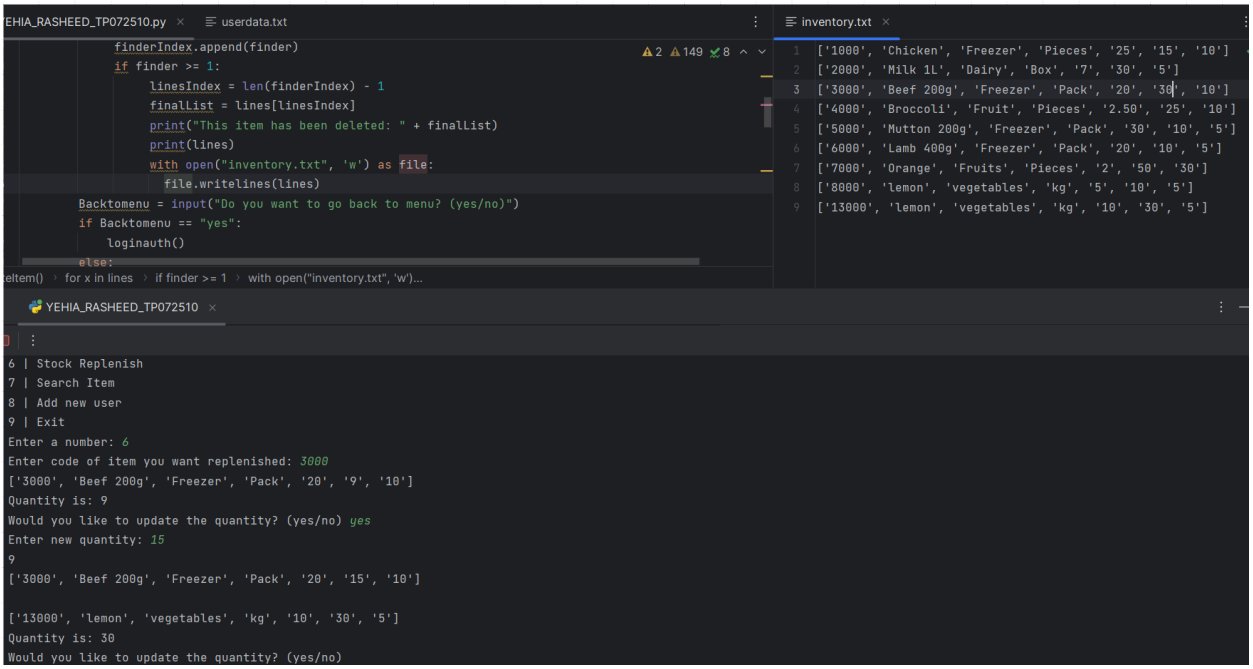
unit: pack

price: 20

no of item: 9

minimum: 10

This shows that there are now one less beef item in the inventory than the required minimum of ten beef items. So, a replacement for this item is anticipated.

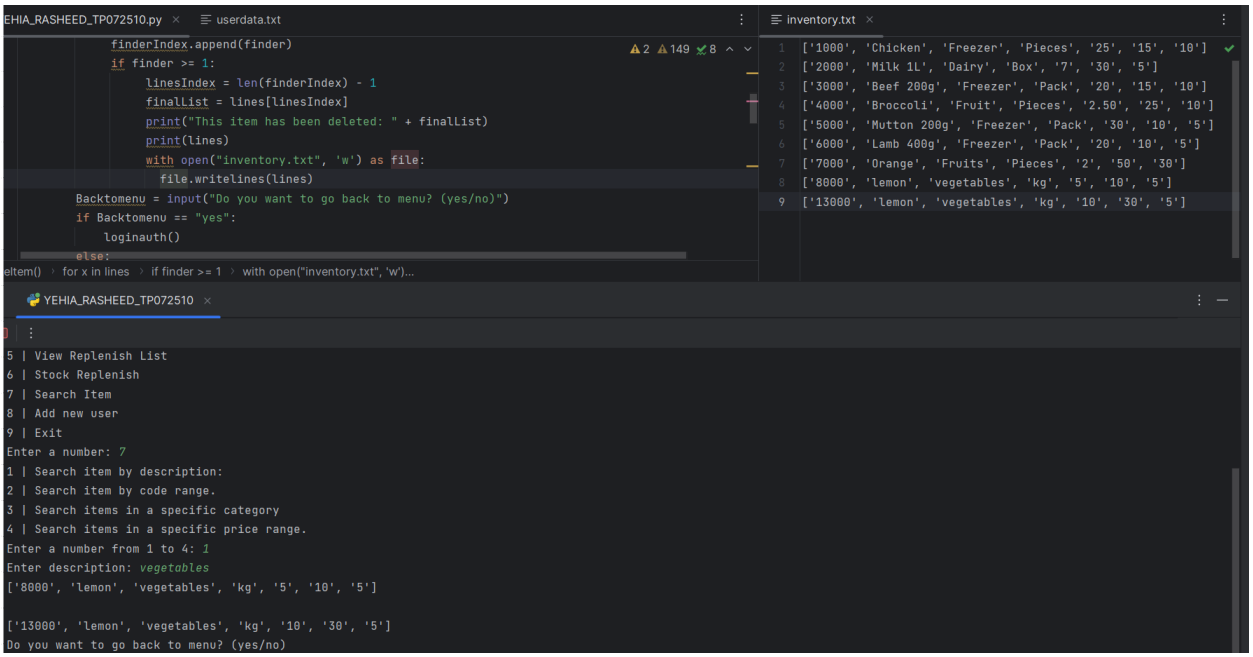


The most recent information on the item, including its amount, is shown by the software. The amount of salt is indicated as 4, indicating that the stock is less than the required minimum of 8.

The application then asks the user to modify the quantity. The user agreed and gave a "yes" response.

When replenishment is necessary, the application alerts the user to enter the new quantity. The user in this instance typed 10.

The computer updates the information on the item and shows the current price, which is now 10.



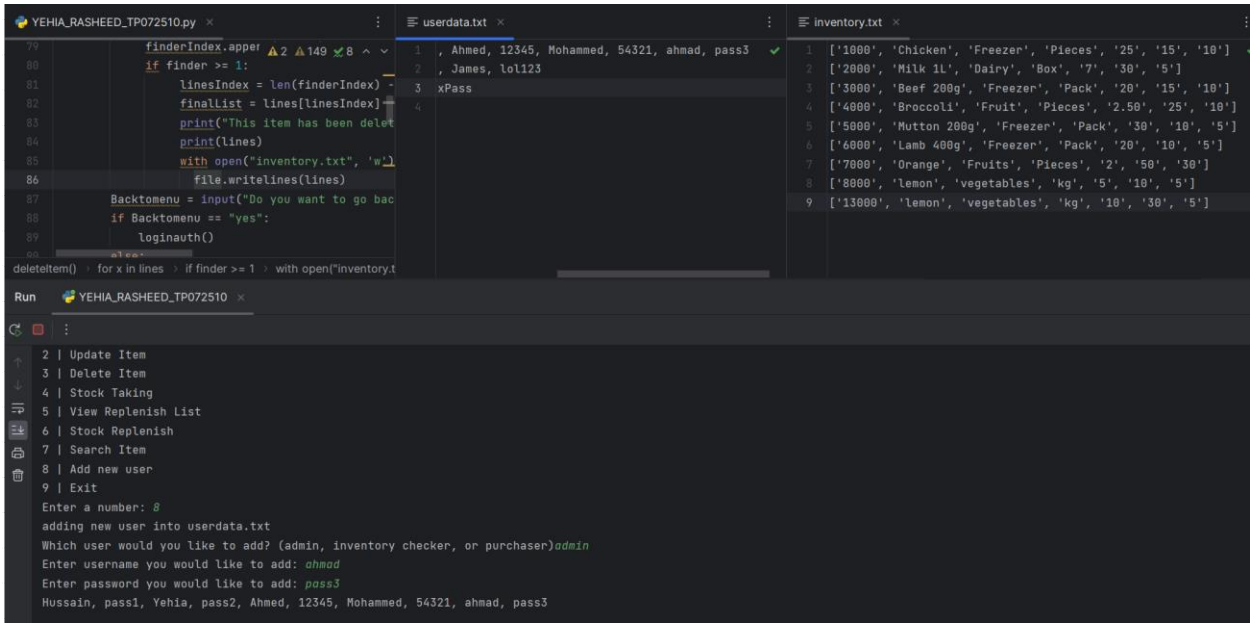
The application offers the user four search options:

Use the items' descriptions or code ranges to do a search.

Look for products in a certain category.

Look for products in a certain price range.

The computer requests a number between 1 and 4 from the user in order to choose one of the search alternatives.



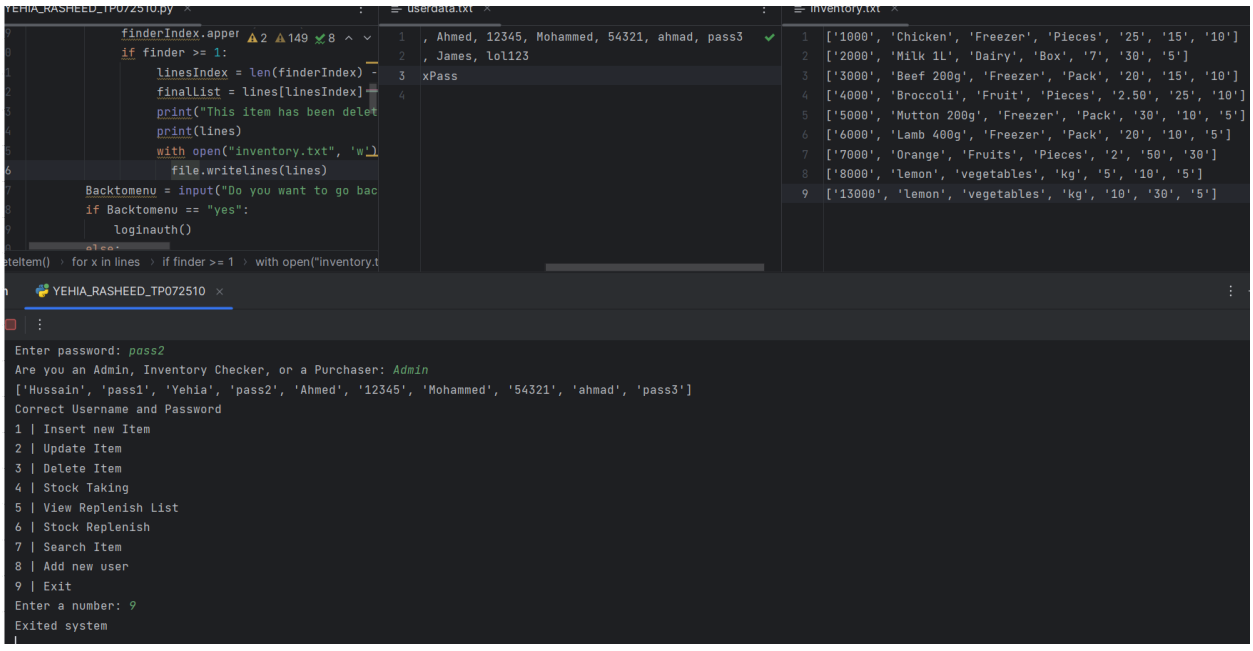
The user will be given the option of adding an administrator, an inventory checker, or a buyer by the computer. The user in this instance decided to add an admin user.

The application then requests the new user's username and password. The user typed "ahmad" for both the username and the password, which was "pass3".

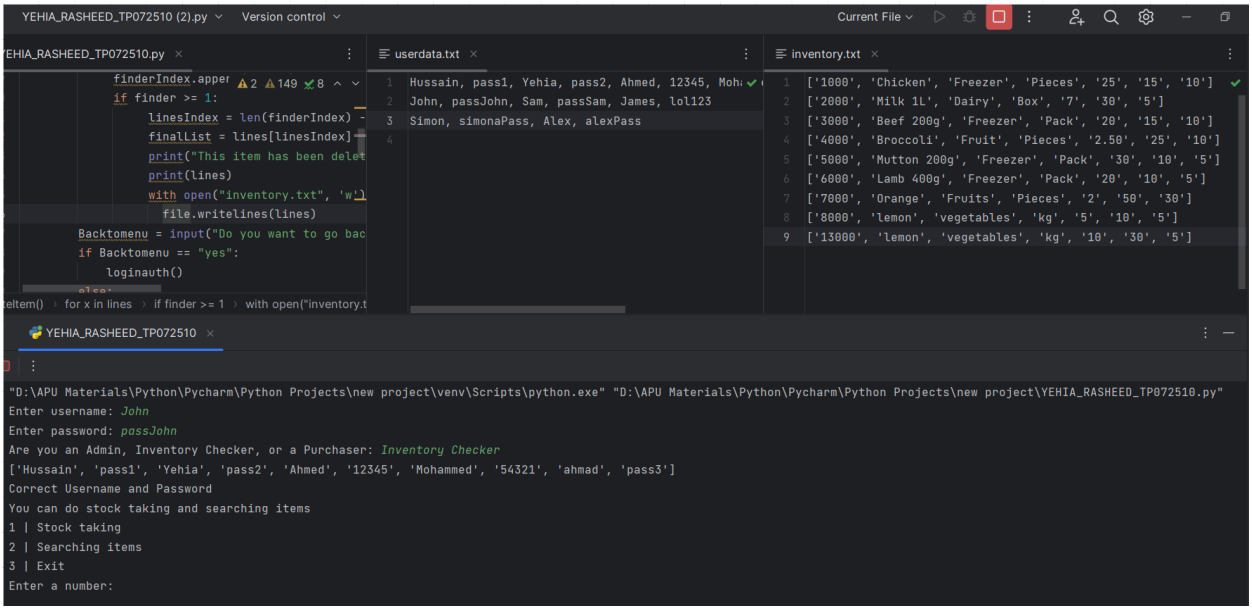
The application then displays a list of users, which includes the just added user, to confirm the inclusion of the new user:

Hussain, Yehia, and 1 through 3 passes, and Hussain

The list of users has been successfully updated with the addition of the new user "ahmad" and password "pass3".



Exiting the grocery store inventory system is indicated by the program's last message, "Exited system," which it displays after completion.

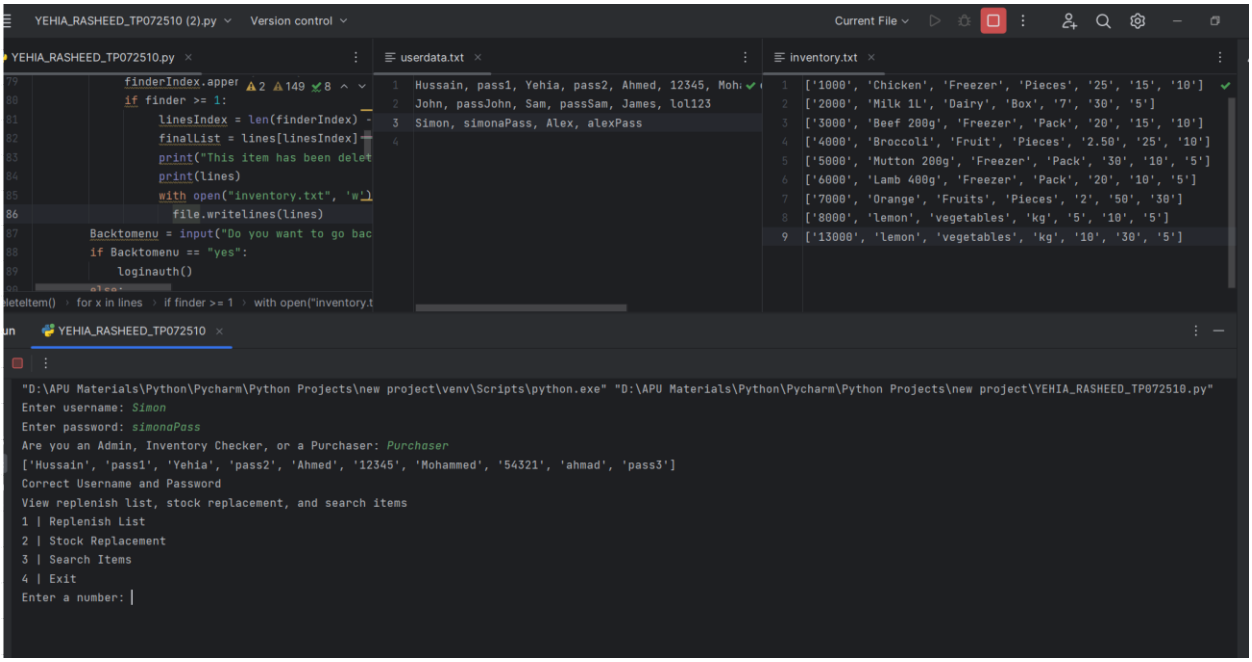


The application verifies the login and password to make sure they are correct.

The following menu selections are then shown by the application:

compiling a list Going from the search result

An related number with the menu item the user wants to choose must be entered.



The application verifies the login and password to make sure they are correct.

The following menu selections are then shown by the application:

Item Exit, Restock List, and Stock Refill Search

An related number with the menu item the user wants to choose must be entered.

Conclusion:

Modern grocery stores employ the Grocery Inventory System for efficient inventory management. Business owners and staff may automate inventory-related operations with this system, which will save on labor costs and guarantee accurate and current inventory data. The system's functions, including item input, updating, and deletion, inventory taking, replenish list building, stock replacement, and item searching, offer comprehensive capacity to satisfy the variety of inventory management requirements.

There are several benefits to the grocery inventory system. It helps store owners manage appropriate stock levels by preventing stockouts or overstocking. Accurate inventory data allows for effective buying decisions, which reduces costs associated with excess inventory or missed sales opportunities. The technology boosts efficiency by automating inventory-related tasks, saving time, and decreasing mistakes. Additionally, it is simpler to manage and evaluate your inventory efficiently when you maintain track of item information such descriptions, classifications, units, price, and numbers. In conclusion, grocery businesses that want to manage their inventories precisely and effectively shouldn't do without the Grocery Inventory System. By leveraging its features and functions, grocery store owners and employees may improve inventory management, customer service, and overall business performance. Any grocery store that wants to improve operational performance and streamline inventory management procedures should take advantage of the system's user-friendly interface.